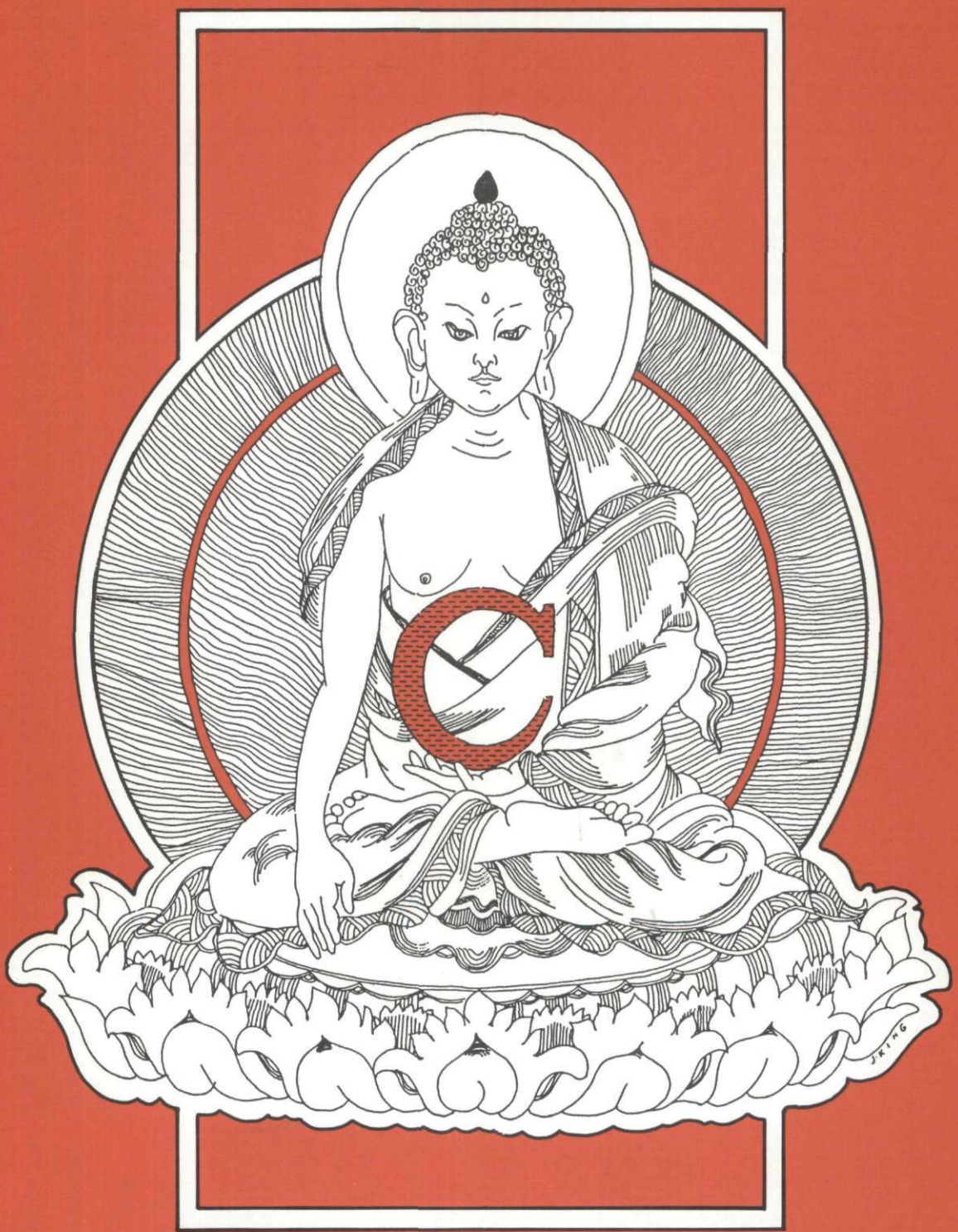
Vol. 3, No. 5 September/October 1985 $4.00

programmer's journal

The Resource Journal for IBM PC Programmers



BULK RATE
U.S. POSTAGE
PAID
EUGENE, OR 97401
PERMIT NO. 668

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Tools for the Programmer from Blaise Computing

Save Up To $130 On These Special Offers!

TOOLS & TOOLS 2

For C or Pascal

For a limited time, pick up both packages and save $50 off our regular list price. The C version comes with libraries for the Lattice, Computer Innovations and Microsoft (version 2.03 and 3.00) compilers. The Pascal version supports IBM and Microsoft Pascal. **$175.**

VIEW MANAGER With Source

All libraries are included. Please specify C or Pascal. Regular $425. Save $130. **$295**

Blaise Computing provides a broad range of fine programming tools for Pascal and C programmers, with libraries designed and engineered for the serious software developer. You get clearly written code that's fully commented so that it can serve both as a model and also be easily modified to grow with your changing needs. Our packages are shipped to you complete with comprehensive manuals, sample programs and source code. None of the programs are copy-protected.

FOR C AND PASCAL PROGRAMMERS:

TOOLS ◇ $125
Extensive string and screen handling, graphics interface and easy creation of program interfaces. Includes all source code.

TOOLS 2 ◇ $100
Memory management, general program control and DOS file support. Interrupt service routine support. Includes all source code.

VIEW MANAGER ◇ $275
General screen management. Create data entry screens that can be easily manipulated from your application program. Block mode data entry and retrieval with fast screen access.

VIEW LIBRARY Source ◇ $150
Source code to the VIEW MANAGER library functions.

ASYNCH MANAGER ◇ $175
Powerful asynchronous communications library providing interrupt driven support for the COM ports. All source code included.

FOR THE TURBO PASCAL PROGRAMMER:

Turbo POWER TOOLS ◇ $99.95
Extensive string support, extended screen and window management, interrupt service routines, program control and memory management, interrupt filters. All source code included.

Turbo ASYNCH ◇ $99.95
Interrupt driven asynchronous communication support callable from Turbo Pascal. ASYNCH is written in assembler and Turbo Pascal with all source code included.

PACKAGES FOR ALL PROGRAMMERS:

EXEC ◇ $95
Program chaining executive. Chain one program from another even if the programs are in different languages. Common data area can be specified. Source code included if you're a registered C TOOLS and C TOOLS 2 user.

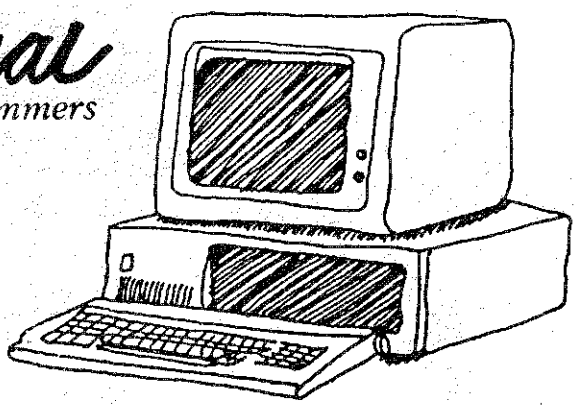
SPARKY ◇ $75
Run-time resident (or stand-alone) scientific, fully programmable, reverse polish notation calculator. No limit on stack size, variables or tape. Includes all standard scientific functions and different base arithmetic.

TO ORDER, call Blaise Computing Inc. at (415) 540-5441

♦ 2034 Blake Street ♦ Berkeley, CA 94704 ♦ (415) 540-5441 ♦

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal
The Resource Journal for IBM PC Programmers



Volume 3, Number 5 September/October 1985

CONTENTS

HOUSEKEEPING 2

PJ FORUM: WINDOWS DONE HERE? by FNE 3
Excerpts from DTACK GROUNDED: The 68000/32081 Systems Journal

CONSULTING: CONSULTANTS AND OTHER FOLK by Paul Barkley 7
Practical considerations get some attention

REDIRECTIONS: C PROGRAMMING BOOKS, PART TWO survey by Frank D. Greco 10
Active criticism of the expert and not-so-expert

C: C CODE EXCERPTS, PART TWO by Ron Moak 17
Advanced routines for more polished programs

ASM: A MENU-DRIVEN PC USING DISPLAY.COM by Michael L. Hoyt 20
Don't spurn a menu for your hard disk system

PASCAL: TURBO PASCAL AND MASM: MAKING THEM WORK TOGETHER by Mark Minasi 28
Get at your library of assembled routines from Turbo

dBASE II: dbase II ROUTINES by Brett Salter 34
Useful routines to ease your way

GETTING USED TO TURBO PASCAL 3.0 review by Tom Swan 38
Admiration and a few complaints

REVIEW: THE TOPVIEW PROGRAMMER'S TOOLKIT by Ron Moak 42
A first look at TopView's Programming Tools

MISC. 45

AD INDEX 46

ISSN# 0747-5861

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSEKEEPING

There are many "Great Moments in History" if your life mainstreams in the high tech environment. As programmers, some of our most satisfying memories come from the knowledge that a business is using the software we have written and that it is helping to complete tasks never before possible. The programs we create for a company make them more money, save them numerous hours, and generate vital reports that corporate decisions are based on.

There are other moments in a programmer's life that are etched in steel. It is Saturday morning, you have the car packed to overflowing with your camping gear, and the phone rings. Somehow you KNOW who is calling (the computer operator), you know what he or she is calling about, and there are occasions when you even know exactly the program at fault and create a visual image of the errant subroutine that has destroyed your weekend. My brother, a computer maintenance sales director in Kansas City, has a story for any occasion. He told me one that duplicates the feeling I've just described. It goes like this:

(Phone conversation)

Bob: Hello.

Doctor: Well, Bob, we got the test results back from the lab. Maybe you better sit down because I have some bad news for you, and then I have some really bad news for you.

Bob: Ok, Doc, give it to me straight.

Doctor: The tests were all positive, and we figure that you only have 24 hours to live.

Bob: Great God! It's a big shock, but I'm glad you were up front. I don't see, however, what you could possibly say that could be worse.

Doctor: Well, you see, we've been trying to get in touch with you since yesterday.

In the computer software business, we are quite familiar with a sinking feeling that has no bottom in sight. I remember the first time that I used a global delete at the operator's console of a big IBM installation. I erased the three files I intended to delete, which saved me all of 20 keystrokes had I decided to do them one at a time. I also erased a 2,000 page inventory report that was not recreateable. It was a "snapshot" report that listed everything in the warehouse on a particular day at a particular time once a month. That episode ended my global delete career on mainframes. Thank goodness a floppy based PC will let you destroy only 360K of information at a whack.

I got a phone call last week from a programmer who needed a copy of the Norton Utilities. His office cleaning lady was vacuuming the hallway while his hard disk was creating a new file. My guess is that the vacuum cleaner ended up with an open random access file, while his hard disk got a number of immaculately clean disk blocks, including the directory. Make sure your PCs are on separate circuits.

Michael Hoyt sent us (and you) what I consider to be the ideal format for a *PJ* article. Take a look for yourself.

— Greg ☐

Publisher:	Elizabeth Oakley
Editor:	Robert Keller
Managing Editor:	Erin O'Connor
Advertising:	Ralph Bentley
Production:	Ralph Bentley
Circulation/Promotion:	Robert Tappan
Cover Design:	Julia King
Illustrations:	Julia King
Contributing Writers:	FNE Paul Barkley Frank D. Greco Michael L. Hoyt Mark Minasi Ron Moak Brett Salter Tom Swan
Founder:	Greg Estes

Published by:	Oakley Publishing Company
Business and Editorial Offices:	Programmer's Journal 1897 Garden Ave. P.O. Box 30160 Eugene, OR 97403 (503) 484-2162
Subscription Inquiries:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07834 (503) 484-2162
Postmaster send address changes to:	*Programmer's Journal* *P.O. Box 3000* *Dept. EE* *Denville, NJ 07834*

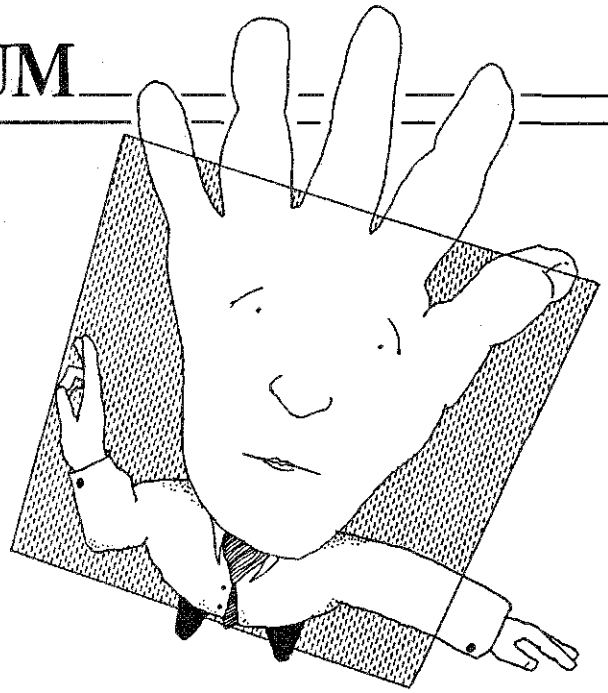
Subscriptions: Bi-monthly rates are $24 U.S. ($30 in Canada and $40 elsewhere)

IBM and Personal Computer are trademarks of International Business Machines;
dBASE II is a trademark of Ashton-Tate.

Published bi-monthly by
Oakley Publishing Company

© 1985

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



WINDOWS DONE HERE?
Excerpts from *DTACK GROUNDED*
The 68000/32081 Systems Journal

by FNE, © 1985 by Digital Acoustics, Inc.

It appears obvious that the future of bit-mapped displays and their associated "raster-ops" is tied to the mass-market acceptance of windows as a user interface technique. And mass-market acceptance (or rejection) of windows is going to be based on mass-psychology whose factors are not yet known. Logical analysis by us or by YOU is going to be totally irrelevant. What we need is careful OBSERVATION: Do you see lots of happy, smiling people forking over money for their bit-mapped personal computers and forking over more money for lots of window-based software to run on their bit-mapped personal computers?

As the politicians would say, all the returns are not in yet. First LISA and now Macintosh were your initial window-machines. LISA is rotting in her grave, and Macintosh sales are, as Apple honchos have admitted on the record, disappointing. But IBM sold off a lot of PCjrs in December with a massive price cut, sales incentives to the individual salespersons AND, get this, a COLOR version of MacDraw! And the forthcoming IBM PC II is widely believed to feature a bit-mapped display. On the other hand, as one wag recently put it, IBM machines are always better BEFORE they are delivered than AFTER!

This leads us to window programs which are optional, that is, the computer user has the option of using them or not (the Mack environment offers no such option). We can discuss such programs in terms of REVIEWSor in terms of SALES.

Reviews of the various window programs are invariably favorable, with the caveat that they tend to eat lots of RAM and they are not, uh, especially swift . . . Sales of (optional) window programs to date have been miserable. It has been estimated that fewer than 20,000 such programs were sold last year. Naturally, as *DATAMATION* reports, some industry guru had predicted sales of 350,000 windowing software packages in 1984! (Shades of UNIX sales predictions?)

Why not just look up the *DATAMATION* article (Mar., p. 48) entitled "BROKEN WINDOWS" and subtitled "TopView, GEM, VisiOn, Desq, and Windows from Microsoft were all introduced with great fanfare more than a year ago. But few are buying." Or, you can read the front-page feature articles on IBM's TopView in two consecutive recent *PC WEEKs*. It seems that TopView has now been released and it has fallen flat on its face!

So the news from the marketplace as of today (this is being typed on Mar. 10) is by and large discouraging for window fans (please excuse small pun). Before we discuss possible reasons why this is so, let us first ask why anybody or any company would invest the necessary enormous sum of money required to develop a window package? (Let us bow our heads for one minute in remembrance of VisiCorp, a large and profitable concern which was utterly destroyed by the cost and ancillary strains of creating VisiOn.)

Well, a windowing package provides a uniform user interface PROVIDED all the software vendors port their application packages to that windowing package (as LOTUS, for instance, most certainly will not). But mainly, windows are SEXY! Who can deny the whiz-bang thrill of watching lots and lots of windows magically appear and move and/or disappear? Who stops to think whether all those windows are USEFUL? Gosh! Gee! Lookitthat!

Personal computer users would do well to consider similar issues with regard to windowing programs before forking over about $500 for an optional program. What are the costs? What are the benefits? IN FACT, WHAT DO WE USE WINDOWS FOR IN THE FIRST PLACE?

PRACTICAL WINDOW STUFF:

Many personal computer industry observers believe that IBM's TopView will be the bellwether of the window-software industry. If TopView, with IBM's money and marketing and prestige behind it, flops, then windowing is dead except for certain cutesy Yuppie-type computers like Mac. Well, TopView has been released, and early indications are that it has flopped BADLY. Why?

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

IBM PCs, unlike Macs, are generally not purchaced as adult toys. They are commonly purchased to do one job and one job only — word processing (WordStar or MultiMate), spreadsheets (Lotus 1-2- 3 or Symphony), communications (Crosstalk). The first problem with TopView is that it takes up nearly 200Kbytes of RAM, we kid you not. Since IBM's BIOS limits the PC to 640K RAM, that can be a problem even for the few folks who have all 640K filled.

WINDOWS ARE SLOW!

The second problem is Lotus 1-2-3 and the very snappy screen response provided by 1-2-3, which bypasses the ROM BIOS and directly addresses the display hardware. 1-2-3 customers have become accustomed to that snappy response, and it is almost totally responsible for the enormous sales advantage enjoyed by Lotus' over such programs as Context's rather leisurely MBA. Lotus is understandably reluctant to degrade the performance of 1-2-3, reducing its (substantial) advantage over its competitors. (Actually, 'reluctant' is not the word: try 'refusal,' as in "Hell, no!")

The third problem is that an important "advantage" of window programs is that they permit rapid switching between two or more applications. We say again, most IBM PC-type personal computers are purchased for ONE PURPOSE AND ONE PURPOSE ONLY! Why pay $500 for a program which eats 200K RAM and slows down your ONE application? Why, indeed?

WINDOWS ARE MEMORY HOGS!

But this does not apply to YOU, dear reader! Heck, no! You use your computer for LOTS of different applications, and you want to switch between them rapidly. Besides, you like windows. So take your IBM PC and load up TopView (kiss 200K goodbye). Now load your first application, Lotus' 1-2-3 (kiss another 300K goodbye if you are using even a moderate number of rows and columns on your spreadsheet). Now load MultiMate. Ooo! We seem to be running a tad short of memory! Let's see, now . . . load Crosstalk, the communications program. Oops! We seem to be out of memory here!

Naturally, the above scenario is a fantasy because Lotus refuses to produce a crippled version of 1-2-3 which will crawl, slowly, under TopView. Are you beginning to see why TopView has not caught on in the mass marketplace?

DO YOU GOT A CHOICE?

The captive window programs, such as the Macintosh operating system, have a better shot at success (you don't got no choice, chum!). But have you seen the latest sales figures for Mac? DRI's GEM has a rosy future if it is the only operating environment which is going to be supported on the Jacintosh, and provided that Jack can find the jack to produce the Jacintosh in mass-market quantities.

But it does appear at this time that the optional window programs are doomed. It will be interesting to see if the bit-mapped display of the forthcoming PC II can turn this situation around.

Please enter my new subscription to

programmer's journal

Bi-Monthly Subscription Rate

	U.S.	Canada (U.S. Funds only)	Foreign
1 year	☐ $24	☐ $30	☐ $40
2 years	☐ $44	☐ $55	☐ $75
3 years	☐ $64	☐ $80	☐ $110

Mail to: P.O. Box 3000 Department EE
Denville, NJ 07834

☐ New Subscription
☐ Renewal

☐ Check enclosed
☐ Bill me
☐ Charge my Bankcard: ☐ VISA ☐ MC
Card No. _____
Expiration date _____

Signature

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Telephone _____

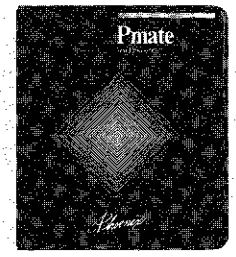
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

One Text Processor Beats Pmate™. The New Pmate.

No other PC/MS-DOS® text processor lets you edit files at the same time that you're compiling a program. Or, call up your editor while inside another application. Or, put long text processing jobs in the background and run other programs while they execute.

No other text processor offers you built-in macros that give you C and FORTRAN language-specific editing features. Brace matching. Movement, deletion, and copy by function. Keyword insertion. Error processing, and many more. Plus, since you get macro source code for all language-specific features, you can modify or add features as you wish.

And, no other text processor lets you use 1-2-3®-like menus, mouse pop-up menus, command-driven operation, or any combination of the above. With Pmate's 170-command macro language, you can program a single key to handle multiple command sequences. Call up other macros or common statements. Pass arguments to macros. Or, set up your own personalized text processing system with all your favorite features.



Plus, you get full DOS 2.x pathname support. 100 numeric variables. A number stack for storing intermediate results. Arithmetic and logical operations. Structured control statements. As well as numerous string, text and document manipulation functions.

And, of course, you get standard Pmate features such as full-screen single-key editing. Automatic disk buffering. Ten auxiliary buffers. Horizontal and vertical scrolling. A "garbage stack" buffer for retrieval of deleted text.

Pmate is priced at $225 and is available for 8086-based micros running MS-DOS. A custom version is available for the IBM® PC, TI Professional,™ Wang Professional,™ and DEC Rainbow.™

Find out more about the new Pmate today! Call (800) 344-7200. In Massachusetts (617) 762-5030. Or, write.

Phoenix Computer Products Corp., 320 Norwood Park South, Norwood, MA 02062

PROGRAMMERS' PFANTASIES™ BY

Phoenix

See us at COMDEX/Fall '85

Pmate and Programmers' Pfantasies are trademarks of Phoenix Computer Products Corporation.
MS-DOS is a trademark of Microsoft Corporation. 1-2-3 is a registered trademark of Lotus Development Corporation.
IBM is a registered trademark of International Business Machines. TI Professional is a trademark of Texas Instruments Incorporated.
Wang Professional is a trademark of Wang Laboratories, Incorporated. DEC Rainbow is a trademark of Digital Research Corporation.

MBASIC Programmers

Save **Time & Money** with

HELP MESSAGE SYSTEM™

Minimize documentation! Create and edit, context sensitive, interactive help and/or tutorial functions. Turns your text information into callable libraries. Overhead to your program is insignificant.

B-PLUS TREE SUB-SYSTEM™

Can use unique or non unique keys. Direct and sequential accessing, variable length keys, multiple non unique keys and and multiple keys per record. Get First, Next, Last or Prior key in sequence. Can access a key field in a file of 32,000 records in less than 1 second!

SCREEN BUILDER SYSTEM™

Using either the built in editor or your word processor, create edit fields of any length anywhere on the screen with full screen editing within each field and full screen movement from field to field. The generated code is ready to insert in your program. Does not make any calls to BIOS. Does not use BLOAD or BSAVE. Bulletproof editing.

EZ SCREEN™

An Assembler language routine embedded in a basic subroutine to perform screen handling and windowing functions when called from a Basic program.

Priced As Low As **$59** Each

30 DAY $ BACK GUARANTEE
NOT COPY-PROTECTED
NO ROYALTY

Any 1 Program	$79	
Any 2 Programs	$138	($69 each)
Any 3 Programs	$198	($66 each)
All 4 Programs	$236	($59 each)

Visa, Mastercard, Checks or C.O.D.
CA residents add 6.5% sales tax
No shipping charges

STERLING CASTLE

702 Washington Street Suite 174
Marina Del Rey, California 90291
(213)306-3020

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCTIVITY UTILITIES

From Opt – Tech Data Processing

NEW **On-Line Help** NEW

A comprehensive utility for adding help windows to your programs. It provides an efficient method for storing help texts in libraries and standard routines for interfacing your programs with these libraries.

Help windows are displayed in a fraction of a second. You have total control over the contents of the window, its position on the screen and the display colors used.

On-Line Help can be interfaced with interpreted Basic and all popular compilers.

On-Line Help includes a Help Library Manager, Library Interface Routines and source code examples. **$149.**

Opt-Tech Sort

High performance sort/merge utility can be used as a stand-alone program or called as a subroutine to many languages.

Supports unlimited filesizes, multiple input files and fixed or variable length records. Many special file types are supported including Btrieve and Dbase. Up to nine sort control fields (ascending or descending), all common data types supported. Output file can be full records, keys or pointers.

Written in assembly language for high performance. Example: 4,000 128 byte records sorted to give key and pointer in 30 seconds. **$99.**

Scroll & Recall

Allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations).

Allows you to easily recall and edit your previously entered DOS commands and data lines without re-typing.

Scroll & Recall is very easy to use. It's a resident utility that's always there when you need it! **$69.**

All programs are fully documented and are compatible with the IBM-PC, XT, AT and compatible machines using the DOS operating system.

Visa, M/C, Check, Money Order,
COD, or PO accepted.
Quantity and Dealer discounts available.

To order or to receive additional information write or call:

Opt-Tech Data Processing
P.O. Box 678 — Zephyr Cove, NV 89448
(702) 588-3737

Another feature story in an issue of *Electronic News* is on DRI, GEM, and bugs in the Intel 80286. DRI is laying off again—10% of its current workforce, this time. It seems that GEM, which is a wonderful windowing program that is designed to take advantage of wonderful bit-mapped graphics, is a severe cash drain. [Let's see, now: VisiOn destroyed VisiCorp, IBM's TopView has bombed, but Microsoft's windowing program has NOT bombed because it is LATE LATE LATE and has not yet been released. Obviously, windows and bit-mapped graphics are wonderful? *Electronic News* notes, "No firm has yet succeeded at marketing GEM-like products."]

In addition to GEM's cash drain, DRI's Concurrent DOS 286 is running late because the Intel 80286 has bugs. You know that fabled built-in memory protection the 80286 has? The memory protection which slows the chip to a walk if switched on? Well, that memory protection and some other stuff DOES NOT WORK! If you will not take our word for this, perhaps you will accept Intel's word? *Electronic News* quotes Intel's George Alexy, marketing manager for high-performance microprocessors, thusly: "There is a legitimate bug in the part. [The memory protection and DOS emulation capabilities] worked in the B-1 step but not in the C-step. We're going to make a change in the E-1 step."

Electronic News further notes, "He said the part's shortcomings were discovered by DRI once Intel had advanced to the D-1 step, in which it attempted to make speed enhancements. Intel is currently in production on C step 80286 . . . . It plans to start producing the version corrected for Concurrent DOS 286 capability in the third quarter. The scheduled shipment date to DRI is 'confidential,' he said."

[Note that Intel is desperately trying to enhance the speed of the 80286, with little success. IBM is still shipping ATs with 6MHz parts. And it was two years ago that AMD was running its "busted tricycle" ads telling us what a hot performer the 10MHz 286 was! Let's see, now: we shipped our first 12.5MHz 68000 board product, using a processor routinely purchased across the counter of Hamilton Electro Sales, on 1 June 1982! Also, we reported recently that the 80286 was bug-ridden and received an indignant letter from an Intel employee, who shall remain anonymous, that "there ain't either any bugs in the 286!" SUUURRE there aren't.]

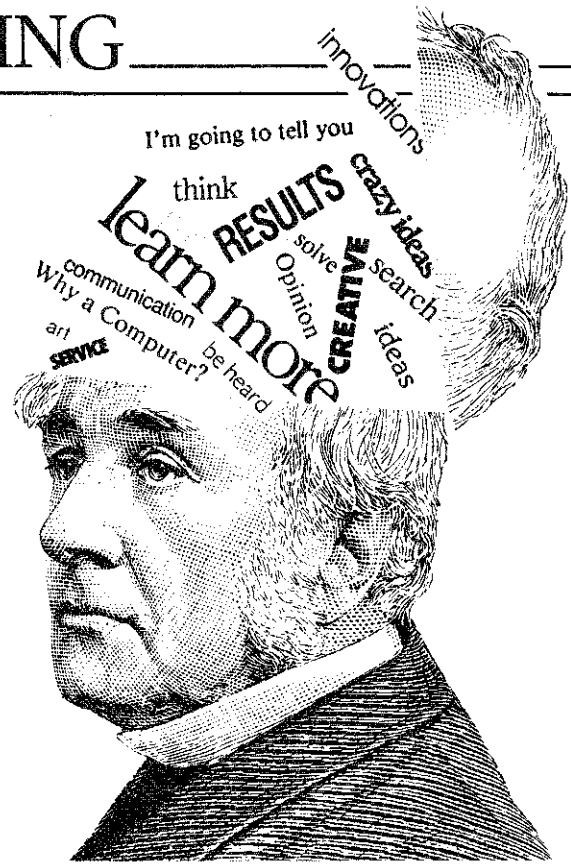
Electronic News' story covers 27 column inches and covers much more detail on DRI's woes and the 80286's bugs than we are reporting here. This issue of *Electronic News* has lots more interesting stuff, such as yet ANOTHER feature story, this one on IBM's becoming its own supplier of hard disks for the AT. It seems lots of disk suppliers have a strong tendency to go belly up, have declined IBM's kind invitation to bid for the business.

Since Victor and Ed have already blown our modesty image, we do not mind pointing out that ALL of the above stories in that issue of *Electronic News* were printed in greater or lesser detail in *DTACK GROUNDED* FIRST! □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CONSULTANTS AND OTHER FOLK

by Paul Barkley



Welcome to a new series of feature articles in *PJ* devoted to software and hardware consultants! I come into contact with a wide variety of computer consultants, and I've noticed that almost all of them are willing to hold forth at great length on almost any topic that you toss out. There seems to be some sort of natural selection process that results in consultants being eager to talk.

Well, I reasoned, since consultants are such a talkative lot, and since what they have to say is almost always interesting, writing a column for consultants should be a piece of cake — as long as I make it a forum for discussion, and not a soapbox for my own views, wonderful though I might find them.

So, what I want to do is throw out a topic, say a few words about it, and then get your reaction to it. My address is at the end of the article.

I'd like to start off by defining what a "computer consultant" is. Unfortunately, this is a word with as many meanings as politicians have promises. First off, we have the "independent computer consultants" — that's how I usually describe myself. I've never met a "dependent consultant" (perhaps that's a consultant hooked on drugs), but the idea is that independent consultants are primarily consultants and not salespersons masquerading as consultants. The latter always come up with (what a surprise) solutions identical to the products that they sell. If you're "independent," that could mean you're ornery, but it should mean that you have no financial stake in your recommendations. Anybody who works for a computer store and has a business card that says "consultant" — isn't. Consultants get paid for their expert time, not for selling products.

Some independent consultants work for large companies, and some are essentially one-man firms. If you accept the meaning of "independent" in the financial sense, the size of the firm has no bearing. Some people insist, however, that only small, somehow "pure" firms can truly be independent. What do you think?

In the narrowest sense of the term, independent computer consultants don't really do any work — they just consult. A consultant talks to clients, pokes his nose into all the nooks and crannies of the client's affairs (in the process making a nuisance of himself), and then goes back to his office to write up a scathingly accurate analysis of the numerous ridiculous business practices of the client, finishing with a checklist of brilliant suggestions for implementing an automated system sure to triple the profits of the client within six months.

If the consultant is really comprehensive, he or she may also advise the client during the acquisition of this wonderful system and earn his or her fees by making life miserable for the hardware and software vendors involved (by acting as the client's advocate).

These are the most visible activities of independent computer consultants (other than constantly marketing their own services, of course) and generally command respectable fees. These are also the activities that have earned consultants of all kinds (even outside the computer field) the popular definition: "A consultant is a person who borrows your watch to tell you what time it is — and then charges you for it."

Those analysis activities are the ones that I personally enjoy most. I don't know anyone who fits that narrow definition, though. Almost all independent consultants wind up doing a considerable amount of custom software development as well— unless they're part of a large enough organization where those tasks can be left to others.

My experience is that I'm almost never called in unless the client already has made two observations:

1. No one has been intelligent enough to publish a comprehensive software package that automates the business of selling poultry medications, gothic mystery novels, and two-week alpine tour packages;

2. The proper methods of merchandising the goods above were perfected in the year 1898 and should be automated EXACTLY in that fashion.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

These curious but all too human observations usually lead the client to give standard automation a fair try by calling in a consultant first, in the vain hope that somewhere such a package exists. When, to no one's surprise, either no packages are found, or the one possible package commits the unforgivable error of printing column A to the left of column B on several reports and has "sort of ugly" screens to boot, the natural tendency is to commission a custom applications program.

Does this sound cynical? My standard statement is, "You don't want custom programming unless it absolutely can't be avoided because it costs too much money, takes too much time, and never works exactly right the first time." After about an hour of explaining why the client doesn't want custom programming, I find that the statement, "Of course you DO get a system that does exactly what you want it to do," usually clinches the sale. I try hard to talk them out of it — really I do — but it's almost always hopeless.

Tell me, why do clients hire *your* services? Why do you think they need you? I left out a few good reasons — tell me *your* experiences.

As we've observed, many independent computer consultants usually wind up doing a little applications programming. When I noticed that 90% of my revenue was coming from programming and 10% from "consulting," I decided that this was more than just a little. In fact, I realized that I had somehow managed to get my business totally skewed from what I set out to do.

I used to think that people who mostly programmed were awfully loose with the English language when they called themselves consultants. After waking up to what I was now doing, though, I began to ask other consultants about the relative proportions of their activities. One firm that now has about twenty people said, "Nah, we don't do any consulting anymore — we just do programming." Could it be that we've stumbled across an unknown evolutionary principle? Is it true that consultants are merely the larval stage of programmers? Do consultants metamorphose into applications programmers with a few, presumably inadvertently exposed to radioactive forces, metamorphosing into systems programmers? Has this happened to you? Do you know someone this has happened to? Do you wish it would happen to you? Are you moving in the opposite direction and devolving from a programmer to a consultant?

Besides "pure" consultants and programmer/consultants, there are a lot of people who seem to me to be contract programmers. Their clients take them on for extended periods, like six months to a year, and they're not involved with much of the design and analysis. Their task is almost identical to that of a regular company programmer, but without any benefits. The client hires them on a half- or full-year contract,

A FULL C COMPILER FOR $49.95

The Eco-C88 C compiler is setting a new standard for price and performance. Compare Eco-C88's performance to compilers costing up to 10 times as much:

	Seive	Fib	Defref	Matrix
Execute	12.1 sec.	43.1 sec.	13.7 sec.	21.3 sec.
Code Size	7782	7754	7772	9120
Compile-link	76 Sec.	77 Sec.	77 Sec.	92 Sec.

Eco-C88 Rel. 2.20, on IBM PC with 2 floppy disks, 256K. Benchmarks from Feb., 1985 **Computer Language.**

Eco-C88 includes:
* All operators and data types (except bit fields)
* Error messages in English with page numbers that reference the **C Programming Guide** - a real plus if you're just getting started in C.
* Over 170 library functions, including color and transcendentals
* New Library functions for treating memory as a file
* User-selectable ASM or OBJ output (no assembler required)
* 8087 support with 8087 sensed at *runtime*
* cc and "mini-make" for easy compiles (with source)
* Fast, efficient code for all IBM-PC, XT, AT and compatibles using MSDOS 2.1 or later.
* Complete user's manual

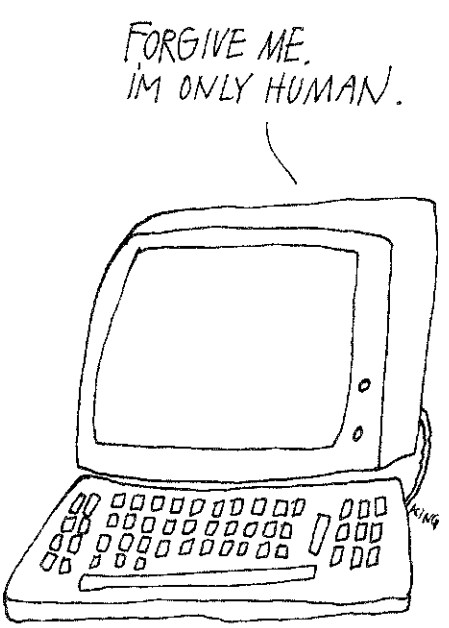
If ordered with the compiler, the C library source code (excluding transcendentals) is $10.00 and the ISAM file handler (as published in the **C Programmer's Library,** Que Corp) in OBJ format is an additional $15.00. Please add $4.00 for shipping and handling. To order, call or write:

MasterCard VISA AMERICAN EXPRESS

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476 • 8:30-4:30

ECOSOFT INC.

Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).



The First A.I. Error Message

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

at a slightly higher rate, because at the end of that time they're disposable, in a way that regular employees are not. If contract programmers want to call themselves consultants, I guess that's all right with me.

Finally, there are people who perform the traditional consulting activities, as opposed to primarily programming, but who also wind up on very long contracts. I know an interesting gentleman who has been on various contracts with the same federal agency for five years now — he still calls himself a consultant, and that's probably an apt description. Perhaps we should call him a "captive consultant." His only defect was to attend the wrong university in Michigan — the one in Ann Arbor. Otherwise, he's as much a consultant as I am.

In the Washington area, and perhaps elsewhere, there's an interesting term used by those who tend toward long term contracts, whether contract programmers or captive consultants. When their contract runs out, they're said to be "on the beach." Consultants who work with a variety of clients are almost never on the beach — they always have a half dozen overlapping projects in progress at any time. The latter are constantly marketing their services and putting projects in the queue, while the former tend to do their marketing sporadically as each long term contract expires. This crisis marketing is almost identical to the job hunting process carried on by an out-of-work employee.

In addition to all the categories above, there are a number of people who perform various consulting activities on a part time basis. Many of them cannot afford to leave their regular employment, and yet they want to explore setting up a business on their own — it's a powerful lure.

I've talked about these various categories because I think it's important to understand that consultants have a wide variety of different views and bring with them conflicting values and needs. Consequently, I expect to see a real variety of viewpoints from *PJ* readers who do consulting.

If you're thinking about becoming a consultant, you should match the type of consulting you expect to do with your personality. Do you want to be the expert that a small firm relies on for guidance, or would you prefer to become part of a major programming team in a large corporation? Can your brain handle multiple tasks running concurrently (think of it as the biological analog of the CPU), or do you prefer to work on one large task at a time?

Topics for the future include marketing your services; how to price your services; contracts necessary to do business; the merits of hardware and software specialization; bridging the gap between consulting and software publishing; tool development strategies for successful applications programming; and anything else of interest that readers want to get into.

Here are two premises for the next article, sure to provoke debate:

1. Very few people can successfully run a computer consulting business from home.

2. Almost everyone who is a part time consultant does both clients and employer a great disservice.

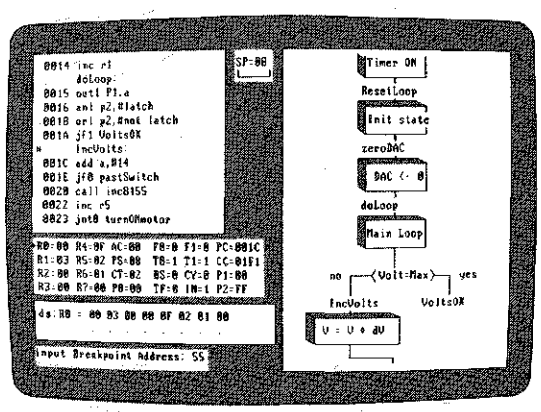
Let's hear the arguments!

Drop me a line:

Paul A. Barkley
The Barbican Company
6845 Elm Street Suite 710
McLean, Virginia 22101-3896

Or contact me on CompuServe at 74776,2526. □

Simulator/Debugger for IBM-PC for 8048 - 8051 - 7000 µCs



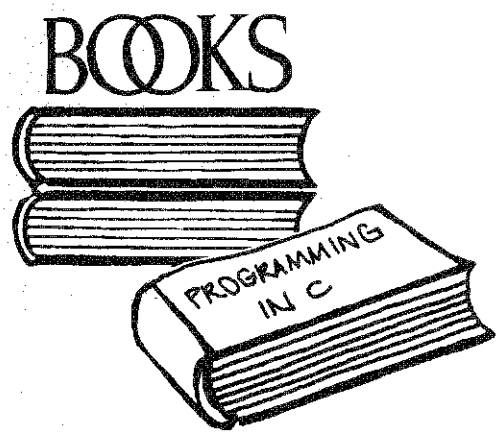
Execute and debug code for popular single chip microcomputers on your IBM PC. Dynamic display with windows for source code, control flow, registers, flags, memory, commands, and more! Set breakpoints, traps, etc. Cross-assembler & EPROM programmers, too.

	8048	8051	7000	Z8	
X-Assembler	✓	✓	✓	✓	$295
EPROM Prog	✓	✓	✓	✓	$245
Simulator	✓	✓	✓		$595

Demo disk and manual for simulator only $39.50

Cybernetic Micro Systems
Box 3000 • San Gregorio, California 94074 U.S.A.
(415) 726-3000 • Telex 171-135 Attn: Cybernetics

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



C PROGRAMMING BOOKS (Part 2)

survey by Frank D. Greco

In this issue of PJ *we publish the balance of Frank D. Greco's comprehensive survey of C programming books. Those of you who missed the first part of this critical review can find it in* PJ, *Volume 3, Number 4, July/August 1985.*

13. *The C Programming Tutor*
— Wortman and Sidebottom — Brady Publ.

This book is mediocre at its best. It tries to be helpful to the beginning C programmer; however, its disorganization is its downfall. In addition to its strangely ordered scheme of topics, the physical layout is eye straining. Instead of typesetting the listings in this book, the authors elected to use a NEC 7710 Spinwriter with a Pica-10 thimble.

Also, the authors' use of illustrations is quite crude. Pictures—I use that term very loosely—are "drawn" with standard ASCII characters! Didn't someone at Brady think of using professional graphics artists? If the authors like to draw, why didn't they at least use a Macintosh or some other bit-mapped machine? The layout is atrocious.

The introduction says some pretty strange things: " . . . you also may have learned there is virtually no other authoritative C text available to you. Even the computer journals and periodicals are short of articles [on C]. Very few have been published concerning this language . . . none offers or claims to offer . . . anything remotely resembling a tutorial on how to write useful C source code." Note that this book was published in 1984; these statements about non-existent C tutorial works are totally misguided.

Listings in the book are extremely contrived. For example, the authors use a macro named FALSE in an "if" statement; e.g.,

```
#define FALSE  0
```

Then later in a program segment, they write the following:

```
if ( isalpha(c) != FALSE )
```

This is very poor style for a C program. In addition, this implies that a programmer should think in BASIC or FORTRAN while coding in the C language. The reason that a programmer selects a particular programming language in the first place is for the language's particular features (such as portability, speed, strong on string manipulation, etc.).

If there is something that makes this book worth purchasing, it is the second part, called Useful Programs. Here the authors give the reader the source code to several utility programs. There is a printer configurator (sends escape sequences to a typical microcomputer printer); a program that counts characters, words, lines, and sentences in a file and displays some statistics; a C program cross referencer; and a C program function call generator (a high level structural look at a C program). There are some interesting techniques in this section of the book, but the authors assume that the reader can keep up with their fast pace. If only the first part of the book were as useful as the second . . .

14. *C Graphics* — Novack and Korites — Kern Publications Inc. Publ.

This is not a book about computer graphics using C as I had hoped. Rather, *C Graphics* is a manual describing a library of functions to perform graphics on the IBM PC that one can buy from the publishers, Kern International, for $28.00. As such, let's look at this work as a manual, not as a book.

The listings in this small manual (84 pages) look like they were photocopied from an old-ribbon Epson dot-matrix printer. Isn't this the age of laser printers and graphics devices? Most of the explanations of the code are fairly clear; however, the bases of the algorithms that the authors use are not brought out in the C language at all. The authors do show you a lot of BASIC code to describe their algorithms . . . sneaky, sneaky.

If you want to learn more about algorithms for graphics use, get the classic graphics text *Fundamentals of Computer Graphics* by Foley and Van Dam, or the new book by Bruce Artwick (author of the Flight Simulator) called *Microcomputer Displays, Graphics, and Animation*. As a manual for the function library that Kern is selling, *C Graphics* is average; as a book explaining computer graphics, it needs a lot of work.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Instant-C™: The Best Value In C Programming Tools

The edit-compile-link-test-debug cycle that takes tens of minutes with compilers and linkers is only seconds with the *Instant-C* interpreter. Yet it runs your programs **50 to 500 times faster** than conventional C interpreters! You get the best of both compilers **and** interpreters. Only *Instant-C* is a complete, integrated environment for creating, testing, and running your programs.

Instant-C gives you **all** of these proven capabilities in one tightly integrated package:

interpreter—*Instant-C* runs your programs faster than some compilers; has direct execution; full K&R

compiler—*Instant-C* can make stand-alone programs

full-screen language editor—shows syntax errors with cursor set to trouble spot

C source debugger—single-step, breakpoints, stack trace, more

run-time checker—validates pointer refs, array bounds, more

C source formatter—save editing time, find logical flaws

standard library with source—for best portability

linker—work with multiple source modules

Lint—extensive compile-time validation

The cheapest available examples of these tools would cost $800 (and they don't even work together). You could spend close to $3000 to get the best product of each kind, but you'd have ten times the complexity, filling megabytes of disk. *Instant-C* is faster: it performs these functions automatically. *Instant-C* is far more than the sum of its parts.

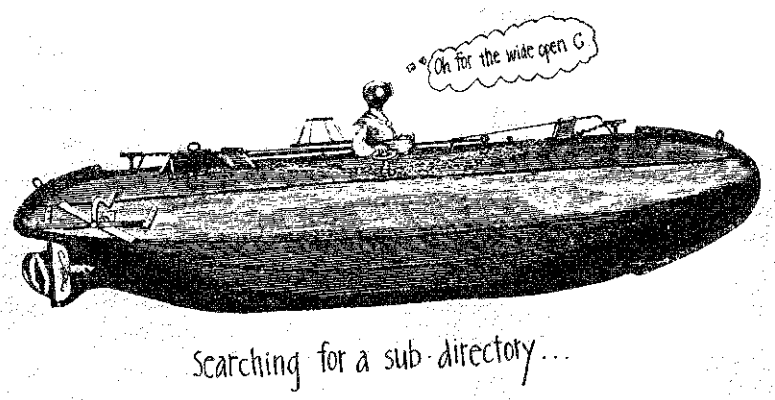
Instant-C is all of these capabilities in one package, fits on a single floppy disk, is full K&R, works on IBM PC's, compatibles, and others under DOS or CP/M-86. It costs only $495.

Instant-C is the best value in C programming tools. Guaranteed, or your money back for any reason in first 31 days.

Rational Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, MA 01760

Instant-C is a trademark of Rational Systems, Inc.



YOU NEED A GOOD LIBRARY

POWER PACKS™

COMPLETE SOURCES NO ROYALTIES

COMPREHENSIVE C Power Packs include over 1000 functions which provide an integrated environment for developing your applications efficiently. "This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
— Dr. Dobb's Journal, July 1984

USEFUL "...can be used as an excellent learning tool for beginning C Programmers..."
—PC User's Group of Colorado, Jan. 1985

FLEXIBLE Most Compilers and all Memory Models supported.

RECOMMENDED "I have no hesitation in recommending it to any programmer interested in producing more applications code, using more of the PC capabilities, in much less time." — Microsystems, Oct. 1984

- **PACK 1: Building Blocks I** $149
  DOS, Keyboard, File, Printer, Video, Async
- **PACK 2: Database** $399
  B-Tree, Virtual Memory, Lists, Variable Records
- **PACK 3: Communications** $149
  Smartmodem™, Xon/Xoff, X-Modem, Modem-7
- **PACK 4: Building Blocks II** $149
  Dates, Textwindows, Menus, Data Compression, Graphics
- **PACK 5: Mathematics I** $99
  Log, Trig, Random, Std Deviation
- **PACK 6: Utilities I (EXE files)** $99
  Arc, Diff, Replace, Scan, Wipe

Master Card/Visa, $7 Shipping, Mass. Sales Tax 5%

ASK FOR FREE DEMO DISKETTE

NOVUM ORGANUM INC.

SOFTWARE HORIZONS INC.

165 Bedford St., Burlington, MA 01803
(617) 273-4711

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

15. *Introduction to C* — P. Chirlian — Matrix Publishers

I had to chuckle a little bit when I read the introduction to this book. Mr. Chirlian, who is from the Electrical Engineering department of Stevens Institute of Technology in New Jersey, explains a programming statement as follows:

"Suppose that we wanted to write a statement in the C programming language corresponding to the following algebraic equation.

$$x = a + b/c$$

The corresponding statement in the C programming language would be:

```
x=a+b/c;
```

Note that this closely resembles the algebraic statement and is easy for programmers to write."

Closely resembles? If it weren't for the terminating semicolon, they would be identical! Also, if the reader truly is a programming novice but knowledgeable in algebra, how would he/she discern this: x = x + 5; ????? Not exactly algebra. With this as a prelude, I felt a little uneasy with Professor Chirlian.

Chirlian, like many other C book authors, makes the usual definition/declaration mistake (page 7): "int a,b,c; This is called a declaration statement." Later on the same page: "The declaration statement . . . causes the compiler to allocate memory locations . . ." If you followed my previous comments on some of the other books, you KNOW what definitions are and what declarations are. And how about "Memory locations in a computer are called words [?]. An integer may require more than one word for its storage." Didn't Kernighan and Ritchie (K&R) specify that an integer takes up whatever is the "natural wordsize" of the cpu (page 34 of K&R)? Perhaps Chirlian is referring to LONG INTS? If he is, his point is certainly not very clear.

Another early example the author uses is the following:

```
    :
  char buf[81];
int quot,rem,a,b,c;
    :
a=atoi(gets(buf));
    :
```

He says (page 16), "We can enter 80 characters . . . . We simply used 81 as an example.[?]" and also "Note that the input is obtained using . . . gets(buf) instead of . . . fgets(buf,80,stdin) . . . . This shorter form is available on most, but not all, C compilers." Well, the author should have pointed out that 81, instead of the usual 80 (characters per line), was used because the programmer should allocate enough storage (chars) to store the string plus one extra byte for the NULL (binary zero). His comment that function 'gets(buf)' is a short form of 'fgets(buf,80,stdin)' is very misleading. First of all, if fgets() is used with stdin, you must include the standard I/O header file (stdio.h). Stdin is a macro that is defined there (stdio.h). Secondly, gets() gets characters up to a newline character, then replaces the newline character with the ASCII NUL character ('\0'); fgets() gets characters up to a newline, then APPENDS the null byte. Consequently, fgets() retains the newline, gets() does not.

If your standard I/O library that came along with your compiler is a little remiss and has only fgets() and does not have gets(), here is a little help:

```
#define MAXLINE     80          /* Adjust to line size */
  :
  :
char * gets(buffer)
char * buffer;
{
     char * start = buffer;     /* save buffer start */

     fgets(buffer, MAXLINE - 1, stdin);  /* Get line */
     while (*buffer++ != '\n')
          ;                     /* Search for newline */
     --buffer = '\0';           /* Replace with NULL  */

     return start;
}
```

If you need the speed and not the memory, you can create a gets() macro using the C preprocessor:

```
#define gets(x)  ( fgets(x,MAXLINE,stdin),
x[ strlen(x)-1 ] = '\0', x)
```

I parenthesized the macro for safety against other operators, and I used the comma operator to force left-to-right evaluation. The last subexpression, just the dummy parameter x, is to simulate the return of a char pointer.

Introduction to C is one of the me-too C books, and its organization could have been a lot better.

16. *The Small-C Handbook* — Hendrix — Reston Publ.

This book is a description of Ron Cain's public domain beauty, Small-C. Small-C is a subset of the C programming language that was initially written in C for the 8080 microprocessor. The source code was first seen in the magazine *Dr. Dobbs' Journal*. This book contains a concise and correct description of the C language in Section 2, but Section 3 is where it shines. Section 3 delineates in detail the innards of the Small-C compiler. Hendrix describes I/O concepts, function call efficiency, code generation, and how to alter the compiler itself. This is a wonderfully lucid attempt to explain how a compiler works; and it succeeds. Since this topic is on the technical side, I would recommend this book to experienced programmers only. The appendices of this book contain the entire source code of the Small-C compiler (written in C); a CP/M disk is available from the author for $25.00. PC-DOS versions exist in the public domain.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

17. *Understanding C* — Hunter — Sybex Publishers

This book, unfortunately, is just another one of those mediocre books on the C language. Frequently, the author relies on many conventions of the CP/M operating system for file handling instead of on the more portable standard C library. If the biggest advantage of the C language is portability, why use CP/M particulars? Also, by using functions that are specific to the BDS C compiler, the author alienates many potential Lattice, C86 or DeSmet C programmers.

Several spelling errors can be found throughout the text; I feel that spelling errors should be minimized in published works. Let's face it, there are spelling checker programs out there.

Some blatant C language mistakes in this book are obvious even to neophyte C hacks: "The backslash indicates that the number following is in octal." This is very confusing. Any C programmer worth his pay will tell you that a number prefaced by a ZERO is considered to be in octal! The backslash indicates an internal escape sequence known to the C compiler (such as '\n' or '\t'). If you want to represent a character in terms of its ASCII numeric representation, you must precede the number with a backslash; otherwise, C will not consider the numeric characters as one ASCII character. What Mr. Hunter is trying to say is that the ASCII numeric representation is assumed by the compiler to be in octal format. Many of you out there may be wondering why the emphasis on octal in C (and UNIX). The reason for octal is purely historic. UNIX and C were originally brought up on early PDP machines, which were octal based; hence the vestigal base 8 anomalies. I understand that the proposed ANSI standard for the C language has hex, octal, and binary character constants. For example,

```
"error\007"    are the characters e-r-r-o-r followed by a
               control G (beep).

"error007"     are the characters e-r-r-o-r-0-0-7
```

Lack of attention to detail is prevalent in other parts of the book. I know I'm nitpicking, but people pay good money for books, and they deserve quality.

18. *A C Reference Manual* — Harbison & Steele — Prentice Hall

This book is an excellent supplement to the Kernighan and Ritchie classic, *The C Programming Language*, on the formal definition of the C language. The title of *this* book matches the contents! That is a surprise, given the state of affairs nowadays. The authors provide a complete description of the C language; they also discuss the differences among various C compilers out on the market. There are few faults (if any) in this book, and I recommend it as a companion reference book to K&R. However, similar to the K&R classic, this is *not* a book from which to learn C programming. It assumes that you know a little (!) about the language already, and that you are a relatively experienced programmer.

This one is highly recommended, especially for those systems programmers out there writing their own compilers. Only nits: indentation of IF-statements; and placement of the braces after WHILEs and IFs is not my cup of tea, but that's only a personal taste of mine.

19. *Programming HALO Graphics in C* — R. Traister — Prentice Hall

I am surprised that a book of this quality has been released to the general public. Simply put: It is terrible. No, it is worse than terrible; it is incorrect. Here's how. The HALO graphics toolkit is a collection of graphics routines for the PC marketplace. This fine graphics library was written by Jan Gombert of Softwriters, Inc. and is copyrighted by Media Cybernetics. The majority of the HALO library of graphics routines typically expect pointers to be passed to them. For example,

```
int x = 0;
initgraphics(&x);  /* Enable 320 X 200 graphics mode */
```

and

```
int x = 160, y = 100;
movabs(&x, &y);              /* Move cursor to 160,100 */
```

Note that the ampersand ("address of" operator) assures us that a pointer to a variable is sent to the HALO function (remember that an address *is* a pointer!). Therefore, we should not say things like:

```
movabs(160,100);
```

because that would be effectively sending the ADDRESSES 160 and 100 to the movabs function. In reality, the movabs function "sees" two values coming from the caller (i.e., 160 and 100) and "thinks" (assumes) that they are pointers. These particular addresses are in low memory (or the low end of a data segment) and typically are either in ROM, part of the operating system, or somewhere in your own data segment (8086) on most machines.

Presumably Softwriters, Inc., the authors of HALO, required that you use pointers so that their graphics routines would be essentially the same when called by FORTRAN or BASIC. These other languages typically pass parameters by address ("passing by reference").

Now here is where the author of this book begins to falter. Mr. Traister makes a SEVERE error when he says (page 45), "If you omit an ampersand, your error will be recognized by the compiler, so it should be easy to go back and debug," and also (page 43), "If the ampersands are omitted, the compiler will display an error message." WRONG! The C language

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

allows for independent compilation (as opposed to Ada's and Modula's separate compilation). Independent compilation, for one thing, implies that the language has NO knowledge about the "types" of function parameters. The compiler will NOT catch these semantic errors. So as far as the compiler knows,

```
movabs(160,100);
```

is entirely correct, which, syntactically speaking, it is.

Now for the major *faux pas*. The author says (page 43) that "we now know that the values of all variables are passed by address and not by value." WRONG! The C language ALWAYS passes parameters by value; passing by reference can be simulated by passing the VALUE of a pointer. Traister continues on his errant ways by suggesting the use of the following construct:

```
movabs( &(x = 160), &(y = 100) );
```

This is syntactically illegal! The expression above says to pass the value of the address of whatever (x = 160) and (y = 100) evaluate to. So behind the scenes , the C compiler will be performing the following:

```
movabs( &160, &100 );
```

This happens because the value of an assignment expression is the value of the right hand member. For example, the expression (x = 160) evaluates to 160. If you notice, the above statement is trying to take the address of the numbers 160 or 100. This is illegal! You cannot take the address of a constant! In C parlance, you can only take the address of an LVALUE, not an RVALUE. Mr. Traister uses this incorrect construct in practically every example throughout the entire book.

When I called the authors of the HALO package (Softwriters, Inc.) and relayed this information, they were astounded that such a gross error slipped through the publisher's editing process.

In the introduction, the author states that he tested all the examples in the book on an IBM PC with the Lattice C Compiler Version 2.0 under PC-DOS 2.1. I find it hard to believe that the Lattice C Compiler allows these syntactical inconsistencies. If these examples do work, this implies that Lattice has some severe problems with lexical analysis of the C language. This, I doubt.

Not that I want to exacerbate the situation, but if you forget these obvious blunders and look at the rest of the book, you will find that there are about 10 black and white half-page illustrations out of a possible 180 full pages of text. A graphics book that contains a 1:18 graphic-page-to-text-page ratio needs no further comment.

Avoid this book, it is full of BLATANT errors.

20. *Programming in C on the IBM PC* — B. Cummings & L. Pollack — Prentice Hall

This is another of Prentice Hall's Microtrend series of books. As with its companion book *UNIX on the IBM PC* by William B. Twitty, the title is very misleading. This book is actually about programming using the C language in general. The amount of this book dedicated to the PC is from page 3 to page 11, where the authors talk about simple DOS commands and mention the PC-DOS 2.x (and 3.x) escape sequences available to a programmer (enabled at boot-time by DEVICE=ANSI.SYS in your CONFIG.SYS file).

The authors end this particular segment on DOS commands by saying, "Other than in Section 2, the DOS Technical Reference contains very little useful information for the C programmer." Although it is somewhat correct that the average C programmer probably will need only the information contained in Section 2 of the DOS 2.x manual, a book on using the C language on the PC should delineate the hooks available to call the operating system or the (non-portable) BIOS directly. The remainder of the DOS manual describes all the DOS interrupts in functional detail; and the IBM Technical Reference (which was not even mentioned in the book) contains a complete listing of the BIOS with comments on available entry points. DOS and BIOS interrupts are not useful information for IBM PC C-language programmers?

How about showing the reader how to call assembler routines from the C language? This has become a popular technique with many programmers lately. The authors make no mention of interfacing with assembler, or any other language for that matter. What pre-processor macros would be useful on the PC? When would a programmer use a macro over a function? How can you produce graphics using the C language on the PC? What is an example of how a programmer can control the PC's display with escape sequences? I know that these concepts introduce non-portability; however, for maximum speed or control one must occasionally bend the rules. Besides, a good programmer should know how to "layer" his/her software to hide the non-portable issues.

Well, that's what the book should have covered. Unfortunately, like the Traister book, this one also makes erroneous statements about the C programming language.

When explaining precedence, the authors say " . . . introduces the concept of precedence, or the order in which expressions are evaluated." WRONG! Precedence and Order of Evaluation are two different concepts. For example, how would the following expression proceed in evaluation?

```
if( (i < MAXLENGTH)  ||  (arr[i++] != NUM) )
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

According to the authors' rules, the second portion of the if test (that is, "arr[i++] != NUM") should be evaluated first because array subscripting and incrementing have higher precedence than any operator in the first expression ("i < MAXLENGTH"). This is incorrect: Although the precedence dictates the "binding," the evaluation of logical ORs and logical ANDs is guaranteed left-to-right.

How about the following C language expression:

```
arr[i] = i++;
```

Does this expression increment the variable "i" after assignment or before the assignment? According to the authors' mistaken beliefs, the array subscripting should be evaluated first, then assignment is made, and finally, the increment of i. The correct answer to this one is that the result is UNDEFINED. In other words, it is compiler dependent. You are relying on the order of evaluation; that is dangerous in practically any programming language.

The authors also explain that the increment and decrement operators (++ and --) exist for simplification of "i = i + 1" or "i += 1" expressions. This is grossly incorrect! The increment/decrement operators were "invented" to gain access to the machine level INC and DEC instructions, which are much faster than any machine level ADD instruction.

On page 74, the authors say, "To accommodate...input/output...C compilers come with a 'standard I/O libarary.' This libarary may be a file called 'stdio.h' that contains a variety of functions..." WRONG! The file "stdio.h" is an ASCII (i.e., human readable) file that typically contains a set of standardized macros or typedegs that insulate the programmer from the details of implementation. The library, which the authors are apparently confused about, is an OBJECT file that is used at LINK time when the needed functions in your program are incorporated in the runnable module.

How about this one on page 87: "A companion to the address is the pointer, which stores the address of another variable." This is extremely misleading! A pointer IS an address! A pointer VARIABLE stores a pointer CONSTANT (i.e., an address).

The authors also confuse variable declaration in the C language with the more correct variable definition. Creating an instance of a variable in the C language is called a data definition, which implies the allocation of memory. Data declaration, on the other hand, is communicating information about variables defined (i.e., created) somewhere else.

Not to nitpick, but the authors explain, " . . . the final function you can perform on the files as a whole is to expunge a file from the disc. unlink(filename) . . ." No other explanations or examples are given to restate or reinforce this confusing sentence. In other (simpler) words, the authors are attempting to convey the method of deleting (or "erasing") a file from the disk. Why should the authors be so obscure?

I could go on, but essentially because there are numerous errors in the book, I would strongly recommend leaving this one alone.

21. *The C Programmer's Handbook* — Thom Hogan — Brady Publ.

Mr. Hogan says in the preface that "You will not learn to program in C here." This is not a C language tutorial; it is more a reference manual for constructs of the language. Each operator, standard library function, and flow-of-control construct has at least a complete page dedicated to listing the functionality, restrictions, and precedence (if applicable) and an example. The book is organized pretty well, and with minimal errors. What Mr. Hogan might have done to make life easier for the programmer is to color code the sections: the operators section one color, the function call section another color, and the flow-of-control section another color (not to mention the appendices). If color was too expensive for the publishers, black markers on the edge of each page could have shown where each section is.

One of the appendices (there are eight) is titled "Assembly Language in C." While this sounds very sexy, it really applies only to the compiler that the author was using, the Software Toolworks c/80 C compiler. It turns out that this compiler's

INTRODUCING
DATALIGHT C

The Datalight C Compiler for MSDOS is a full C with all K&R constructs, including bitfields, plus the version 7 extensions. Other features of the compiler are:

$60

- Produces object files (.obj) so just the MSDOS linker is required.
- Floating point performed with 8087 or automatic software floating.
- Over 100 compact library functions with source.
- Compatible with the Lattice C compiler.
- Runs on IBM-PC, and compatibles, running MSDOS 2.0 or later.
- Complete, easy-to-read users' manual with index.
- Highly optimized code for production quality programs.

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

Outside USA add $10 shipping. Washington State residents add 7.9% sales tax. VISA and MasterCard accepted.

IBM-PC, a trademark of IBM; MSDOS, a trademark of Microsoft Corp.; Lattice C, a trademark of Lattice Corp.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

preprocessor has a directive called #asm (like some other compilers) that says "assume that everything from now on is in assembler." This directive is turned off by a #endasm. This type of directive is usually available on compilers that subscribe to the UNIX philosophy of compiling C to assembler, then assembling to machine code. #asm/#endasm merely protect a section of code from being compiled to assembler (it already is in assembler!).

Another appendix is the bibliography, which is one of the highlights of this handbook. Hogan lists several books that deal directly with the C language and some that discuss it peripherally to another topic (i.e., UNIX). He also lists some 50 magazine articles dealing with this elegant language.

Some of my students tell me that they appreciated the layout of this book, but still had a hard time finding certain items. Some of my students say that they thought the book was very helpful. I say it's a jump ball.

22. *C: An Advanced Introduction* — Narain Gehani — Computer Science Press

23. *Advanced C: Food for the Educated Palate* — N. Gehani — Computer Science Press

24. *C for Personal Computers: IBM PC, AT&T PC 6300, and Compatibles* — Narain Gehani — Computer Science Press

These three books by Narain Gehani of AT&T Bell Laboratories are almost identical. Mr. Gehani's version of the loaves and fishes can probably be attributed to reading too much James Martin. *C: An Advanced Introduction* is supposedly written with the student in mind, *Advanced C: Food for the Educated Palate* is for the professional programmer rather than the student, and *C for Personal Computers* is for the small non-UNIX environment (apparently Gehani took all the references to UNIX out of his original work here). A close friend who bought both *Introduction* and *Food for the Educated Palate* bitterly complained that the books are virtually identical! Luckily, I received a flyer from Computer Science Press stating that the books are versions of each other and bought just the *Educated Palate* version.

Funny games aside, Narain Gehani is well known for several books and papers on concurrent programming and the Ada programming language/environment. He is well versed in the C language. He is the co-designer of the new Concurrent C language, which has concurrency constructs built into the language *a la* Modula II and Ada. Although some parts of *Educated Palate* have a strong bias towards UNIX, there is enough useful information to warrant a purchase. The *Advanced Introduction* version is listed at $29.95, the other two versions are listed at $19.95. I am told that the more expensive student version (*Advanced Introduction*) has many examples of useful C code. I personally found the *Educated Palate* version enlightening. One of the items discussed thoroughly in the "Some C Tools" chapter is the UNIX based 'Make' facility. The newly proposed ANSI standard for the C language (2 pages) and C++ are also discussed.

B. Dalton bookstores have the *Educated Palate* version listed in the top 10 computer books of the week. It is listed as Number 8 "with a bullet."

25. *The C Programmer's Handbook* — Bolsky & Matthews — AT&T Bell Laboratories

The title of this book reminds me of the day that AT&T announced their grand networking scheme on January 1, 1984. That day they announced Net 1, the network that networks all. Something that it almost netted was a lawsuit from communications experts Ungermann-Bass, who already had a product called Net 1. Since marketing did not do their homework very well, the phone company was subsequently forced to call their networking system Net 1000. Well, the Brady Publishers have a book by Thom Hogan called *The C Programmer's Handbook*, which I have had for a while. Will Ma Bell be forced to change the name of this book to *The C Programmer's Notebook*?

Anyway, this book amounts to an extended "cheat sheet" on the C language. Despite what the authors say in the beginning of this small notebook (70 pages) about its being an introduction and a reference to the language, it is just a reference. The authors use color and boldfacing to clearly define topics and important rules in the language. They also describe many of the functions in the standard library. Both the types of the parameters and the types of the functions are given in a clear fashion (if you know how to read UNIX manuals). Unfortunately, the authors fail to tell you more vital information about the return values of these functions.

I know that fgets() returns a pointer to a char, but what char? I know that strcmp() returns an int, but what value is this int? If only the authors had described the return values in a little bit more detail, I would strongly suggest picking this one up as a programming aid for beginning C programmers. Microcomputer C programmers should be aware that there are a few functions listed in this handbook (er, notebook) that exist only under a UNIX environment. Unfortunately, these functions are NOT listed as being "UNIX-only." □

Frank D. Greco can be reached
% Programmer's Journal
P.O. Box 30160
Eugene, Oregon 97403

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE EXCERPTS, PART TWO

by Ron Moak

C CODE EXCERPTS

In the last issue of *PJ* (Volume 3, Number 4, July/August 1985), we published the first part of Ron Moak's assembly language functions that manipulate the screen and can be linked into C programs. Part One provided some basic functions to manipulate the cursor, initialize the screen, change mode, and read characters from the screen buffer.

Part Two contains more advanced functions for removing and restoring blocks of characters from and to the screen, printing lines with attributes, and scrolling selected portions of the screen.

These functions were written to be assembled with IBM's or Microsoft's Macro Assembler. The resulting object file may be linked directly into C code compiled with the Lattice compiler using the small memory model. Slight modifications to the source code will be required to support different compilers or memory models.

If you would like to make use of these functions or of those published in the previous issue of *PJ*, you may get the complete source, plus the object file, for $15.00. Please send a diskette with return postage and check to the following address:

Ronald Moak
Forest Computer Consultants
447 Brookside Dr.
Eugene, OR 97405

```
comment *-------------------< HEADER >-----------------------------------

FILE:     SCRIO.ASM

TYPE:     ASM - Functions

PURPOSE:     Low level assembly language functions designed to interface
     with Lattice C.  These functions deal with screen input
     output.

COPYRIGHT:  Forest Computer Consultants (C) 1985

NOTES:     These functions maybe linked into the Lattice C programs
     compiled to use the small memory module.

FUNCTIONS:  scratt ( row,col,length,attribute )
     scrget ( srow,scol,erow,ecol,buffer )
     scrinit( )
     scrprt ( row,col,string,len,attr )
     scrput ( srow,scol,erow,ecol,buffer )
     scrsrl ( srow,scol,erow,ecol,lines,attr )


----------------------------< EQUATES >--------------------------*

;  Video Functions

VIDEO       EQU  10H      ;VIDEO INTERUPT

SCR_MODE       EQU  0H      ;set screen mode
CUR_SET        EQU  1H      ;set cursor type
CUR_LOCATE     EQU  2H      ;set cursor position
CUR_GET        EQU  3H      ;get cursor type & location
READLP      EQU  4H      ;reading light pen position
SCR_PAGE       EQU  5H      ;select active page
SCROLL_UP      EQU  6H      ;Requests scroll up
SCROLL_DN      EQU  7H      ;scroll screen nown
CH_READ        EQU  8H      ;read a character from screen
CHA_WRITE      EQU  9H      ;write char and attributes
CH_WRITE       EQU  AH      ;write character only
SCR_STAT       EQU  15      ;find current screen status

;  Constants

ATTRIBUTE      EQU  7H      ;standerd attribute character
BLANK       EQU  20H      ;blank character
SEGM_MONO      EQU  0B000H     ;Mono card segment
SEGM_COLOR     EQU  0B800H     ;Color card segment
ADDR_MONO      EQU  03BAH      ;Address of mono card
ADDR_COLOR     EQU  03DAH      ;Address of color card
SCR_CHAR       EQU  2000       ;Number of character stored on screen
CUR_LINE       EQU  1      ;two line cursor
CUR_HALF       EQU  2      ;half block cursor
CUR_BLOCK      EQU  3      ;full block cursor
CUR_COL_LIN    EQU  0607H      ;two line cursorcolor screen
CUR_COL_HLF    EQU  0407H      ;half block cursor color screen
CUR_COL_BLK    EQU  0007H      ;full block cursor color screen
CUR_MON_LIN    EQU  0A0DH      ;two line cursormono  screen
CUR_MON_HLF    EQU  070DH      ;half block cursor mono  screen
CUR_MON_BLK    EQU  000DH      ;full block cursor mono  screen

SCR_MONO       EQU  7      ;screen type 80 col mono
SCR_COLOR      EQU  3      ;screen type 80 col color
SCR_BW      EQU  2      ;screen type 80 col black & white
NUMCOL      EQU  50H      ;number of columns on the screen
NUMROW      EQU  19H      ;number of rows on the screen

;-------------------------< DATA SEGMENT >---------------------------

dgroup  group   data
data  segment para public 'data'
  public  SCR_TYPE
  assume  ds:dgroup

SCR_ROW        DB      ?     ;the row of the window
SCR_COL        DB      ?     ;the col of the window
SCR_TYPE       DB      ?     ;the type of screen
SCR_BUFF       DW      ?     ;the address of current screen buffer
ADD_6845       DW      ?     ;the address of the 6845 status port

data ends

;-------------------------< PROGAM SEGMENT >-------------------------

pgroup group prog
prog segment byte public 'prog'
 public  curget,curput,curset,scratt,scrcls,screol,scrget,scrinit
 public  scrmode,scrprt,scrput,scrread,scrsrl
 assume  cs:pgroup


;-----------------------------------------------------------------
; namescratt -- sets character attributes
;
; synopisscratt(row,col,count,attribute);
;   short   row;      row (0-24)
;   short   col;      column (0-79)
;   short   count;
;   char    attr;
;
; descriptionThis function sets the attribute for count characters
;starting at row,col. Cursor position is not changed.
;
;-----------------------------------------------------------------

scratt proc   near
 push   bp
 mov   bp,sp
 push   es
 push   di
```

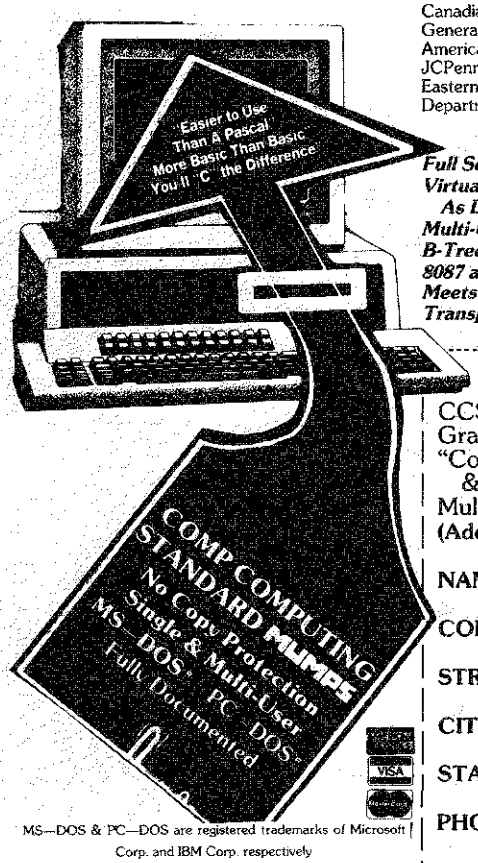

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
 mov   ah,[bp]+4   ;get position
 mov   al,[bp]+6   ;
 mov   SCR_ROW,ah  ;
 mov   SCR_COL,al  ;

 call   CAL_OFFSET   ;es:di has screen address
 inc   di    ;increment for attribute byte
 mov   ax,[bp]+10   ;AX=attr
 mov   cx,[bp]+8   ;CX=count
 cld
sa1:
 stosb   ;move attribute
 inc   di   ;increment past character
 loop   sa1   ;loop and decrement cx by 1

 pop   di
 pop   es
 pop   bp
 ret
scratt endp



;---------------------------------------------------------------------
; namescrget -- get a block of attr/char pairs on the screen
;
; typepublic
;
; synopsisscrget( srow,scol,erow,ecol,buffer );
;   short   srow;      row of upper left  corner
;   short   scol;      col of upper left  corner
;   short   erow;      row of lower right corner
;   short   ecol;      col of lower right corner
;   char    *buffer;   address to place window contents
;
; descriptionCopies from an area of the screen marked by the cooridnates
;srow,scol to erow,ecol into an area in memory pointed to
;by the pointer in the variable buffer.
;
;---------------------------------------------------------------------

scrget proc   near
 push   bp
 mov   bp,sp
 push   bx
 push   si
 push   di

 mov   ax,[bp]+8   ; calculate the number of rows
 sub   ax,[bp]+4   ;
 inc   ax   ;
 mov   [bp]+8,ax   ;

 mov   ax,[bp]+10   ; calculate the number of columns
 sub   ax,[bp]+6   ;
 inc   ax   ;
 mov   [bp]+10,ax   ;
 xor   ax,ax

 push   es
 mov   ah,[bp]+4    ;get position
 mov   al,[bp]+6    ;
 mov   SCR_ROW,ah    ;
 mov   SCR_COL,al    ;

 call   CAL_OFFSET    ;es:di has screen address
 push   ds
 push   es
 pop   ds
 pop   es    ;switch es and ds
 mov   si,di    ;ds:si now has screen address
 mov   di,[bp]+12    ;es:di now has destination
 mov   cx,[bp]+8    ;get number of rows
 cld
sg1:
 push   cx
 push   si    ;save si for later
 mov   cx,[bp]+10    ;get number of columns
sg2:
 in   al,dx    ;get status
 test   al,1     ;is it low?
 jnz   sg2    ;wait until it is
 cli    ;no more interrupts
sg3:
 in   al,dx    ;get status
 test   al,1     ;is it high?
 jz   sg3    ;wait for retrace

 movsw    ;write attr/char
 sti    ;interrupts back on
 loop   sg2

 pop   si
 add   si,NUMCOL    ;go to next row
 add   si,NUMCOL    ;again for attributes
 pop   cx
 loop   sg1

 push   es
 pop   ds    ;restore data segment
 pop   es

 pop   di
 pop   si
 pop   bx
 pop   bp
 ret
scrget endp

;---------------------------------------------------------------------
; namescrinit -- screen initialization routine
;
; typepublic
;
; synopsisscrinit(  );
;
; descriptioninitialization routine that is called at the beginning
;of the program to setup initial values
;
;---------------------------------------------------------------------

scrinit  proc   near
 push   bx

 mov   ADD_6845,ADDR_MONO    ;load values
 mov   SCR_BUFF,SEGM_MONO    ;for mono
 mov   SCR_TYPE,SCR_MONO    ;all done
 mov   ah,SCR_STAT    ;get screen mode
 int   VIDEO    ;
 cmp   al,SCR_MONO    ;test mono
 je   siout

 mov   ADD_6845,ADDR_COLOR    ;load values
 mov   SCR_BUFF,SEGM_COLOR    ;for color
 mov   SCR_TYPE,SCR_COLOR    ;
 cmp   al,SCR_COLOR    ;if color screen then
 je   siout    ;all done

 mov   SCR_TYPE,SCR_BW    ;load for black & white
 cmp   al,SCR_BW    ;if b&w then all done
 je   siout

 mov   al,SCR_BW    ;force mode to 80 bw
 mov   ah,SCR_MODE    ;
 int   VIDEO    ;
siout:
 pop   bx
 ret
scrinit  endp


;---------------------------------------------------------------------
; namescrprt -- put a string to the screen with specified
;  attribute
;
; typepublic
;
; synopsisscrprt( row,col,string,len,attr );
;   short   row;      row of string write
;   short   col;      col of string write
;   char    *string;  pointer to string
;   short   len;      length of the string being printed
;   short   attr;     attribute of characters printed
;
; descriptionprints a string at position (row,col) with each
;character displayed at given attribute
;
;---------------------------------------------------------------------

scrprt proc   near
 push   bp
 mov   bp,sp
 push   es
 push   di

 mov   ah,[bp]+4   ;get position
 mov   al,[bp]+6   ;
 mov   SCR_ROW,ah   ;
 mov   SCR_COL,al   ;
 call   CAL_OFFSET   ;es:di has screen address

 mov   cx,[bp]+10   ;length of string to be printed
 mov   si,[bp]+8   ;get the pointer to the string
 mov   ax,[bp]+12   ;get attribute byte
 mov   ah,al   ;place attribute in ah
spjmp:
 mov   al,[si]   ;get string character
 stosw   ;move attribute
 inc   si   ;increment string counter
 loop   spjmp   ;loop and decrement cx by 1

 pop   di   ;restore registers
 pop   es
 pop   bp
 ret
scrprt endp
;---------------------------------------------------------------------
; name          scrput -- put a block of attr/char pairs on the screen
;
; type          public
;
; synopsis      scrput( srow,scol,erow,ecol,buffer );
;                  short   srow;      row of upper left  corner
;                  short   scol;      col of upper left  corner
;                  short   erow;      row of lower right corner
;                  short   erow;      col of lower right corner
;                  char    *buffer    address to place window contents
;
; description   Copies into an area of the screen marked by the cooridnates
;               srow,scol to erow,ecol from an area in memory pointed to
;               by the pointer in the variable buffer.
;
;---------------------------------------------------------------------

scrput proc   near
 push   bp
 mov   bp,sp
 push   bx   ;save registers
 push   si
 push   di

 push   es
 mov   ax,[bp]+8   ; calculate the number of rows
 sub   ax,[bp]+4   ;
 inc   ax   ;
 mov   [bp]+8,ax   ;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
mov   ax,[bp]+10   ; calculate the number of columns
sub   ax,[bp]+6    ;
inc   ax     ;
mov   [bp]+10,ax   ;
xor   ax,ax

mov   ah,[bp]+4    ;get position
mov   al,[bp]+6    ;
mov   SCR_ROW,ah   ;
mov   SCR_COL,al   ;

call   CAL_OFFSET    ;es:di has screen address
mov   si,[bp]+12   ;get the pointer to the string
mov   cx,[bp]+8    ;get number of rows
cld
sp1:
 push   cx
 push   di    ;save di for later
 mov   cx,[bp]+10    ;get number of columns
sp2:
 in    al,dx    ;get status
 test   al,1     ;is it low?
 jnz   sp2    ;wait until it is
 cli    ;no more interrupts
sp3:
 in    al,dx    ;get status
 test   al,1     ;is it high?
 jz    sp3    ;wait for retrace

 movsw    ;write attr/char
 sti    ;interrupts back on
 loop   sp2

 pop   di
 add   di,NUMCOL    ;go to next row
 add   di,NUMCOL    ;again for attributes
 pop   cx
 loop   sp1

 pop   es
 pop   di
 pop   si
 pop   bx
 pop   bp
 ret
scrput endp


;-----------------------------------------------------------------
; namescrsrl -- scroll the specified window
;
; typepublic
;
; synopsisscrsrl ( srow,scol,erow,ecol,lines,attr )
;    short srow,scol;    row & column of top left
;    short erow,ecol;    row & column of bottom right
;    short lines;        number of lines to scroll
;       > 0 scroll up
;       < 0 scroll down
;       = 0 scroll entire window
;    short attr;         attribute of blanked lines
;
; descriptionThis function scrolls the specified window
;either up or down by the number of lines
;specified.  The blanked lines or given the
;specified attribute.
;
;-----------------------------------------------------------------

scrsrl proc   near
 push   bp
 mov   bp,sp
 mov   ax,[bp]+12  ;get nlines
 or    ax,ax
 mov   ah,SCROLL_UP    ;scroll up function
 jns   ss1   ;if nlines .ge. 0
 inc   ah   ;scroll down function
 neg   al   ;scroll down al lines
ss1:
 mov   ch,[bp]+4   ;get upper left row
 mov   cl,[bp]+6   ;get upper left column
 mov   dh,[bp]+8   ;get lower right row
 mov   dl,[bp]+10   ;get lower right column
 mov   bh,[bp]+14   ;get attribute
 xor   bl,bl
 int   VIDEO   ;BIOS video call
 pop   bp
 ret
scrsrl endp


;---------------------< INTERNAL ROUTINES >-------------------------

;-----------------------------------------------------------------
; name        CAL_OFFSET
;
; type        internal
;
; description  calculates to offset into the screen of the upper
;        left hand corner of the window
;
;-----------------------------------------------------------------

CAL_OFFSET  proc    near
     mov      al,SCR_ROW    ;rows into al
     mov      dl,NUMCOL    ;columns per row into bl
     mul      dl    ;multiply rows by columns
     add      al,SCR_COL    ;add the current column
     adc      ah,0h    ;correct for carry
     shl      ax,1    ;times 2 for attribute bytes
     mov      di,ax    ;offset into screen buffer
     mov      es,SCR_BUFF     ;es:di now ready for writing
     mov      dx,ADD_6845     ;dx points to status port
     ret
CAL_OFFSET  endp

prog    ends
     end
```

□



MS—DOS & PC—DOS are registered trademarks of Microsoft Corp. and IBM Corp. respectively

Shell Oil
Canadian Government
General Motors
American Airlines
JCPenney
Eastern Airlines
Department of Defense

Dow Chemical
Veterans Administration
Hewlett-Packard
Chase-Manhattan Bank
Coca Cola
Johnson's Wax
Exxon

THESE GIANTS KNOW THE SECRET OF PROGRAMMING PRODUCTIVITY . . FOR $59.95 . . . BE A GIANT!

COMP COMPUTING STANDARD MUMPS WILL AT LEAST DOUBLE YOUR PRODUCTIVITY

CCSM Offers These Advantges:

Full Screen Editor
Virtual Memory (Routines & Variables May Be As Large As A Disk)
Multi-User - Up to 12
B-Tree File Structure
8087 and BCD Support Included
Meets or Exceeds 1984 ANSI Standards
Transportable From Micro to Mini to Mainframe

SPECIAL OFFER
"Cookbook of MUMPS & Diskette"
200 Pages of Programmers Techniques & Routines
ONLY $9.95 * Save $15.00!
Offer Ends 12 / 31 / 85

CCSM 1 - (800) 257-8052 Continental U.S.

	Qty.	
CCSM Single User System	____	$59.95
Graphics Add-On	____	49.95
"Cookbook of MUMPS & Diskette"	____	24.95*
Multi-User System	____	450.00

(Add $2.00 to each order for shipping)

*When Purchased With CCSM

Guidance Software Inc.
P.O. Box 5362
Kingwood, TX 77325
(713) 360-3561

NAME _____

COMPANY _____

STREET _____

CITY _____

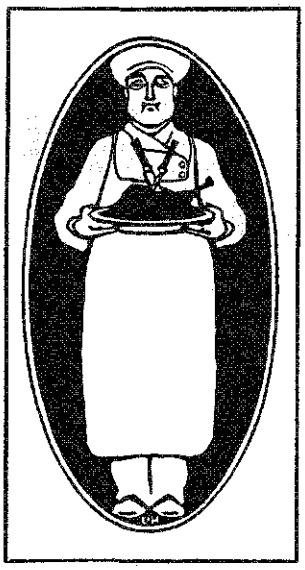
STATE _____ **ZIP** _____

PHONE _____

TOTAL AMOUNT DUE
$ _____

☐ Check Enclosed
☐ Credit Card
☐ Amex ☐ Visa ☐ MC
Credit Card Number ____
_____
Exp. Date _____

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



a menu-driven p.c. using DISPLAY.COM

A MENU-DRIVEN PC USING DISPLAY.COM

by Michael Hoyt

Oh, I fought it for a long time. I had the idea that a menu-driven system was for the novice, the uninitiated, the beginner. Well, I admit it. I was wrong! Now that I've converted, I'm amazed at the time it saves in key strokes. And I no longer have to keep a copy of the hard disk tree structure in my head just so that I can find an application program.

A LITTLE HISTORY

I was one of those who bought an IBM PC way back when there was only 64K memory on the system board, and I could never bring myself to get rid of it just because it was outdated by the XT and hard disks. Instead, I purchased a third-party hard disk and added it to my system along with memory expansion boards, extra ports, clock calendars—the works. The one thing I still have to get is "realistic" backup for the hard disk, either another hard disk (used only for backup) or a tape.

Because of the lack of backup hardware and the fact that I'm basically lazy, I keep application programs on my hard disk only. (There's no way I'm going through a backup procedure to floppy diskettes, even once a week.) That way, I have everything on the hard disk backed up by the original application program diskettes: always. Using such a setup, I don't need to backup. If the hard disk has problems, I've already got everything on diskette.

WHY A MENU SYSTEM

After I'd used the hard disk for a while, it was apparent that I could make the system menu driven. That way, I could eliminate the hassle of remembering where all the application programs were hidden on the disk. Let's see now, was the word processor I use for assembly programs in the \WORD-PRO directory or in the\PROGRAM directory? And all those key strokes to start an application buried way down in the fifth level of sub-directories. How many times does a person have to type a 38-character command line to get it right anyway? I'm sure I hold the record! Well, you get the point.

THE PUBLIC DOMAIN SOLUTION

I'm a great believer in public domain and "freeware" or "userware" (or whatever else you want to call it) software. I admit that I use some commercial products (quite a few, in fact), but when it comes to "utility" programs, I think the best and most useful are often found in the public domain and "user-supported" groups.

So that's where I looked for a menu program. There are several available, some good and some, well, not so good. But, I have this thing about public domain software. If I'm going to use it, I want the source code so that I also know how it works. Too curious I guess. Maybe there's a menu system out there that includes the source code, but I couldn't find it. Next option: Buy a commercial product. No thanks. I've already spent too much money on this computer. Next option: Write one. Well, now, that's something I can do and is exactly what I did.

THE PRELIMINARIES

Designing the system was the first step. I wanted to display a menu on the screen, then press only one key to pick the application. The three functions that needed handling were

Display the Menu
Get the User's Choice (key stroke)
Branch to the appropriate application

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

To begin, I looked at what was available in the public domain so that all I would have to do was write a program that filled in the gaps. After I'd done a little searching, it was obvious that I would either have to use the TYPE command or write my own display routine. Since TYPE is so slow, I decided that writing my own was the answer.

The public domain gave better results for software that returned key stroke codes. ASK and QUERY both allow a "Yes/No" response from the user. GETDIGIT is a bit more flexible since it returns the code of any digit pressed by the user. But best of all was ERRLVL which returns the code of any key pressed—any, that is, but function keys. Granted, I was writing this program for my own use and wasn't going to use the function keys, but I planned to release the program to the public domain so wanted it to be as flexible as possible for others to use. I wanted to return codes from any key, including function keys. Again, it was obvious that my program would also need a routine to handle the return of key strokes codes.

That left only the third function, branching to the appropriate application. I found nothing in the public domain for this. (Maybe I looked in the wrong places.) But I knew from experience that writing routines to handle this could be formidable and quite complicated. Besides, I wanted to keep the program simple enough for use by just about anyone. The solution? Batch files!! I know, I know. They're as slow as molasses in January, but they work. And if you put them on a ram-disk, the speed improves (a little).

After a bit of investigation and decision making, I knew what I had to do: Write a program that displays a menu on the screen then returns the code of the user's key stroke to a batch file.

DISPLAY.COM

As in most of my assembly programs, DISPLAY.COM has one main procedure that calls separate modules (subroutines), each generally handling only a single function. Some of my modules do more than one task (if each task is similar in nature), but as a rule I try to stay away from multi-function modules. Single-function modules make it simpler when it's time for debugging.

DISPLAY.COM begins with a check for the currently running version of DOS. If it's not 2.0 or later, the procedure displays an error message and exits. I figure that by now everyone should be using at least DOS 2.0. If they're not, they should be! The procedure then goes on to clear the screen and disable the cursor. Notice that to make the cursor disappear, I call the ROM_BIOS routine for placing the cursor. There are other ways to disable the cursor, but I like this method of placing the cursor off in space somewhere past the edge of display memory. (I call it "spacing the cursor.") I can then call the same module later (with different parameters) to recover the cursor.

A check is then made for the display adapter currently in use. So far this is pretty standard stuff that any program has to do.

The next portion of code may not seem so straightforward if you don't often make use of the PSP (Program Segment Prefix). The PSP is 100 hex bytes of information built by DOS when it loads a program into memory. It contains lots of useful information including the location of the passed environment, any characters entered after the command name, the amount of free memory available in the program's segment, etc. Although Dos 2.0 (and later) lets programmers ignore the PSP for the most part, it's still useful in certain instances.

Set_Block_Size makes use of the information about free memory, just to make sure that there's enough room available to load the screen menu. That information is gotten from a word of storage at offset 6 (Available_Memory) from the beginning of the PSP. Once we know how much room is available in the segment, it's a simple matter to subtract the number of bytes DISPLAY.COM uses for its code and arrive at the number of bytes left for the menu we wish to display. The result is stored in memory as Block_Size. Generally the amount of available memory will be significantly greater than that needed to load the menu text, but it never hurts to check.

Next we have to make sure the user has indeed included the name of a file to be displayed, and if so, get its name. A call to Get_Filename takes care of it for us. Get_filename also makes use of the PSP by retrieving the characters included on the command line. If all goes well, this will be the name of the file we are to display, not just a user typo. The first byte of the DTA (Disk Transfer Address) contains the number of bytes included on the command line. By loading that byte into the [AL] register and ORing it, we quickly discover if any input was included. If input is present, use of LODSB and STOSB quickly move the characters we want into a special storage location called Filename.

Once we have Filename,the file needs to be opened before anything else. Open_Input does that, returning a file handle in [BP]. [AX] also has the handle but will soon get destroyed. Many programmers would put the file handle in memory storage for later use, but I prefer to keep as much information in registers as possible. Even though it may strain my brain to keep track of each register, I like the payoff in speed.

Obviously the next step is to read the file into memory where we can perform whatever manipulation is necessary for this procedure. By using the DOS function call 3f, "read from a file or device," we use very little code to read the file into our memory buffer. It's at this point that we make sure

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

there is an EOF (End-Of-File) marker placed at the end of the file we want to display. Most text files will already have this marker, but it's best to be sure. Besides, it's "good form," just like the call to Close_Input. We don't actually have to close the file since we haven't changed it in any way, but it's a good habit.

DISPLAYING THE MENU

Now that we've finished the preliminaries, it's time to actually put the menu on the screen. The module that performs that task is Display_Text. The routine is entered with [CX] set to the maximum number of characters we will display (File_Size), [DX] pointing to the status port of the display adapter currently being used (Port), [DI] pointing to the first byte of display memory, [SI] pointing to the beginning of our text file (Buffer), and [AH] holding the normal display attribute.

Since I was going to write the display routine anyway, it made sense to allow a way for the menu designer to change display attributes. Such a feature makes the menu more "readable" by allowing the use of high-lighted characters. I wanted the program to run on either the monochrome or color adapters. That restricts the usable character attributes to high-intensity, reverse, blinking, underline (blue on the color card), and of course normal attributes. Including this little "extra" feature gives the menu designer another tool for directing the user's attention and therefore makes DISPLAY.COM more useful.

Display speed was an important consideration in writing this program in the first place; several things are done here to optimize speed. At first glance it looks like Check_Character should be coded as a called module rather than as part of Display_Text. But when speed is considered, the choice becomes clear. A JUMP instruction is faster than a CALL: about 7 clocks faster. Those ticks of the clock are saved each time we display a character. If we have a screen-full, that's a lot of time!

The second way speed is increased is by structuring the CMP (compare) and J? (conditional jump) instructions so that a single CMP allows elimination of more than one possibility. (For more information on this technique, see "Printer Control from a Batch File," *Programmer's Journal*, Volume 3, Number 3.)

The third way to increase speed is to place each character directly into the display adapter's memory rather than use the ROM_BIOS routine. But since we could be placing the character in the memory of a Color/Graphics card, we have to be careful not to cause "snow." Tacky! This means we have to check the status of the adapter before placing each character, which is what happens in the program loop between Display_Character and End_Of_File. But if you compare the code in this loop with that used by most programmers, you will see that I have replaced

```
TEST AL,01 - JZ (or JNZ) label
```

with

```
ROR AL,01 - JC (or JNC) label.
```

This is also done to increase speed. The ROR instruction is faster than TEST by one clock. Granted, that's not much, but when you add up the clock ticks for each character displayed, it's substantial.

The fourth way to achieve speed is through the use of SHR and SHL beginning at Expand_Tab. I've seen most programmers use a divide instruction (DIV or IDIV) in similar situations. I hate divide instructions and use them only when I have to. They take at least 80 clocks and sometimes over 200. Even the "best case" divide instruction (80 clocks) can't compare in speed to 4 SHL, 1 INC and 4 SHR instructions (18 clocks).

I had fun coding Display_Text. Deciding how to keep track of the current location in the display buffer without accessing the ROM_BIOS memory storage locations was an intriguing job. Using the commented listing, you should be able to figure out just how it's done.

GETTING THE KEY STROKE

I did resort to ROM_BIOS routines for retrieving the user's key stroke. There's nothing special here, other than checking for special keys. If a special key is pressed (i.e., function key with or without Alt, Shift or Ctrl keys and the other Alt key combinations), the keys scan code is retrieved from [AH] and placed in [AL], where it can be passed back to DOS.

After clearing the screen and retrieving the cursor, the procedure passes the code of the key stroke to DOS using the function call 4c. This function allows passing what the DOS manual calls an "error code" when we exit a procedure. Actually we can pass any byte and use it however we want. Notice that if DISPLAY.COM is exited because of an error, the code passed is "ff." We hope this won't happen, but it's there if you need it.

If you look at a table of codes generated by different key strokes, you'll notice that DISPLAY.COM will return the same code to DOS for some function key combinations as it will for certain "standard" key strokes. It's not likely to cause you any trouble, but be advised.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Even though it was designed to be used with text files, DISPLAY.COM will place any file on the screen of your system — even if it's not a text file. Although it displays only a single screen-full of text, it can also be used to take a quick look at the first "page" of any text file. When you design a text file for use as a menu with DISPLAY.COM, you have the choice of most of the 255 ASCII characters. Only those reserved for use as attribute flags can't be displayed. Still, that leaves all the standard ASCII characters and all the characters in the Extended ASCII Set. That should be enough to design a really nice looking menu.

THE MENU

Since DISPLAY.COM will allow branching from a batch file, your menu can be designed for redisplay after you leave each application program. Of course, if you don't like simplicity, you could have lots of menus to display depending on how or where you return from the application program to the batch file. I go for simplicity in this case.

As I stated previously, I keep only application programs on my hard disk. I keep them in directories (and/or subdirectories) so that I can recall where I put them. (Remember, I set up the hard disk before I went to this menu system. If I had to start over, using this system, it's very likely I wouldn't even bother with directories.) Since the number of application programs I play with is truly large, and space here is limited, I've taken the liberty of showing only portions of my MENU, etc.

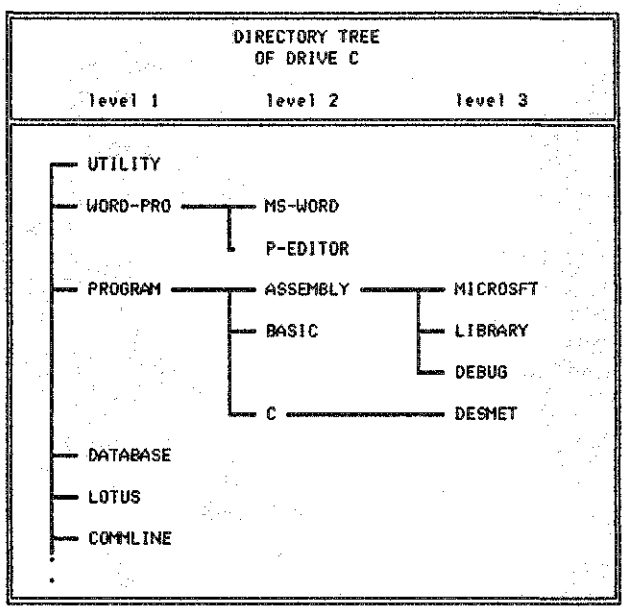


Figure 1

Figure 1 is a partial picture of the TREE structure I use and is included so that you can more easily understand the batch file used to direct the system.

MENU.BAT

```
:begin
d:
e:display e:menu
echo off
if errorlevel 255 goto begin
if errorlevel 82 goto begin
if errorlevel 81 goto Qmodem
if errorlevel 80 goto begin
if errorlevel 79 goto begin
if errorlevel 78 goto begin
if errorlevel 77 goto Word
if errorlevel 76 goto Lotus
if errorlevel 75 goto begin
if errorlevel 74 goto begin
if errorlevel 73 goto begin
if errorlevel 72 goto begin
if errorlevel 71 goto begin
if errorlevel 70 goto begin
if errorlevel 69 goto End
if errorlevel 68 goto Database
if errorlevel 67 goto C
if errorlevel 66 goto Basic
if errorlevel 65 goto Assembly
goto begin

:Assembly
d:
cd c:\program\assembly\microsft
copy c:\word-pro\p-editor\pe.*
dir b:
command
del d:pe.*
goto begin

:Basic
cd c:\program\basic
c:basica
goto begin

C:
cd c:\program\c\desmet24\compiler
copy c:\word-pro\p-editor\pe.*
dir b:
command
del d:pe.*
goto begin

:Database
copy b:baselist.* d:
c:
cd \database\pc-file
pc-file
copy d:baselist.* b:
del d:baselist.*
goto begin

:Lotus
c:
cd \lotus
lotus
goto begin

:Word
c:
cd \word-pro\ms-word
word
goto begin

:Qmodem
c:
cd \commline
qmodem
goto begin

:End
echo on
```

Figure 2

Figure 2 is a much shortened version of my batch file. Notice that the batch file checks the error level in descending order. That order is forced by the fact that comparison is done on a "greater than or equal to" basis. You might also take note that if you check for key strokes that are not consecutive (i.e., 87,86,85,84 . . .), you need to include lines similar to ones I've used to return to the beginning of the batch file. Otherwise, an errant key stroke will cause branching to the label assigned to the next lower ASCII code.

I'll let you figure out the batch file on your own. It's really rather straightforward. Any help you need can be gleaned from the DOS manual. It will help, though, if you know that I use two ram drives. D: is my working drive, and E: is where I copy MENU, MENU.BAT, and DISPLAY.COM when I boot my system. If you are a programmer in any language, you shouldn't have any trouble understanding the branching used in the batch file from the information supplied in Figure 1 (the TREE structure), Figure 2 (the batch file) and figure 3 (the MENU).

```
                    Application Menu
                    ~~~~~~~~~~~~~~~~

      Word Processor                    Data Base (PC File III)

  Assembly Programming (Masm)           Qmodem (communications)

        Lotus 1-2-3                       Basic Programming

      End Menu Control                  C Programming (Desmet)

Press the letter key corresponding to the 1st letter of your choice.
```

Figure 3

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

One thing I've done with this batch file and menu system is make use of COMMAND.COM as a secondary command processor when doing either Assembly or C programming. This allows me to have easy access to DOS and my collection of utility programs. To speed up the loading of COMMAND.COM (and subsequent reloading after overwrites of the command processor), I copy COMMAND.COM to a ram drive. I use an AUTOEXEC.BAT file to do all this for me. It includes the following lines:

```
set comspec=E:COMMAND.COM
...
...
e:
copy c:command.com
copy c:display.com
copy c:menu.*
menu
```

(Notice the use of capital letters in setting COMSPEC. If you don't use upper case, DOS will not change the ENVIRONMENT to your specifications.) Then, when I'm finished, I just type "EXIT" to return to the menu system and the batch file, MENU.BAT.

PROGRAM AVAILABILITY

DISPLAY.COM, DISPLAY.ASM,and a set of instructions is in the public domain (for non-commercial use only) and should be available on many bulletin boards by the time you read this article. Look for a library file called DISPLAY.LBR. If you can't find it, I'll be happy to supply you with a copy on a new diskette for a $10.00 handling fee.

Michael Hoyt
RR 2, Box 65
Solsberry, IN 47459

ARTIFICIAL INTELLIGENCE Programming:

Learn Fast, Experiment,

with the language chosen by JAPAN.

PROLOG-86™

is "Standard" with Tutorials, samples of

- **an Expert System**
- **a Natural Language**

and other sample programs

1 or 2 pages of Prolog would take 10+ pages in C or PASCAL - a different way of thinking. Become familiar in **one evening**.

Intro Price: $125 for CPM-86, MSDOS. Plus $3 Shipping.

Full Refund if not satisfied in first 30 days. **Call 800-821-2492**

Solution Systems
335-J Washington Street
Norwell, MA 02061
617-659-1571

THE PROGRAMMER'S SHOP™

helps compare evaluate and find products. Get answers.

SERVICE: FREE LITERATURE
One free call covers all programmer's software. Ask for a "Packet" on: "AI", BASIC, C, COBOL, Debuggers, Editors, FORTH, FORTRAN, Libraries, PASCAL, UNIX/PC or 30 "addons" for "C"

ARTIFICIAL INTELLIGENCE
ARITY PROLOG - full, debug, to ASM & C, 16 meg use MSDOS $495
EXSYS - Expert System dev't. Full RAM, serious PCDOS $275
GC LISP - "COMMON LISP", rich, tutorial, co-routines PCDOS Call
PROLOG-86 - Learn fast, Standard, tutorials, samples MSDOS $125

C PROGRAMMING
C-terp by Gimpel, full K&R, .OBJ, ASM inter., 8087. MSDOS $255
INSTANT C - interactive development - Edit, Source Debug, run. Edit to Run - 3 Secs. MSDOS $445
"INTRODUCING C" - Interactive C. 500 page tutorial. PCDOS $ 95

Recent Discovery
LISP-86 - "Common" subset, tutorial, editor, PP, trace. Best to learn. All MSDOS. **Only $ 95**

FORTRAN PROGRAMMING
MacFORTRAN - full '77, '66 option. toolbox, debugger, 128K or 512K, ASM-out option MAC $349
RM/FORTRAN - Full '77, BIG ARRAYS. 8087, optimize, back trace, debug MSDOS $459

SUPPORT PRODUCTS
BRIEF Programmer's Editor - undo, windows, reconfigurable, macro programs, powerful PCDOS $195
PLINK - 86 is a program-independent overlay Linker to 32 levels for all MS Languages. C86 and Lattice. $315
Periscope Debugger - load after bombs, sym, 2 screen PCDOS $269
$95 OR LESS for choice of C, ADA, APL, Forth, Modula, others. Call $ 95

Feature
dbVISTA FILE SYSTEM - full indexing, plus optional record types, pointers. Source, no royalties. LAT, C86, MSC, MSDOS $450

Call for a catalog and *solid value*
800-421-8006
THE PROGRAMMER'S SHOP™
128-J Rockland Street, Hanover, MA 02339
Mass. 800-442-8070 or 617-826-7531

New: National Accounts Division 800-446-1185
Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and POs. All formats available. UNIX is a trademark of Bell Labs.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
page    60,132
name    display
title   display a text menu and return code of the keystroke

;;;
```

```
        push    dx                      ;save message pointer on stack
        mov     dx,184f                 ;(25,80) lower right corner of screen
        xor     cx,cx                   ;(0 , 0) upper left corner of screen
        call    Clear_Screen            ;clear the screen to normal attribute
        xor     dx,dx                   ;(0 , 0) set location for cursor
        call    Place_Cursor            ;get the cursor back from nowhere land
        pop     dx                      ;recover the message pointer
        mov     ah,09                   ;set to display the message string
        int     21                      ;use the DOS function call
        mov     al,0ff                  ;set [AL] to show an error condition
        jmp     short Oppa              ;return to DOS and pass the error code
Display endp

page ;================================ CALLED MODULES ================================

;*****************************************************************************
; This routine clears the screen by using the ROM_BIOS "scroll-up" routine.
;
;       INPUT   -       [CX] = upper left corner of display
;                       [DX] = lower right corner
;
;       OUTPUT  -       section of screen is cleared
;
;       CHANGED -       [BX]
;                       [SI], [DI], [BP] flags - destroyed by VIDEO ROM_BIOS
;*****************************************************************************
Clear_Screen    proc near
        mov     ax,0600                 ;set to scroll-up
        mov     bh,07                   ;set the display attribute to normal
        int     10                      ;use the ROM_BIOS routine
        ret                             ;done, so return
Clear_Screen    endp

;*****************************************************************************
;  This routine places the cursor by using the ROM_BIOS routine.
;
;       INPUT   -       [DX] - has the location to place the cursor
;
;       OUTPUT  -       cursor is placed
;
;       CHANGED -       [BX]
;                       [SI], [DI], [BP] flags - destroyed by VIDEO ROM_BIOS
;*****************************************************************************
Place_Cursor    proc near
        mov     ah,02                   ;set to request cursor placement
        xor     bx,bx                   ;set to display page zero
        int     10                      ;use the ROM_BIOS routine
        ret                             ;done, so return
Place_Cursor    endp

page ;*****************************************************************************
;  This routine checks for a filename included on the command line, and saves
;  the name in storage if found.  If not, it returns the error condition by
;  setting the carry flag.
;
;       INPUT   -       none
;
;       OUTPUT  -       filename saved or carry flag set
;
;       CHANGED -       [AX], [SI], [DI], flags
;*****************************************************************************
Get_Filename    proc near
        mov     si,offset Dta           ;point to the command line buffer
        mov     di,offset Filename      ;point to our filename storage
        lodsb                           ;see if command line has any input
        or      al,al                   ;is there a size indicator ?
        jz      Input_Error             ;no, so indicate an error
Scan_Input:
        lodsb                           ;pass over leading blanks
        cmp     al,Cr                   ;have we found a carriage return ?
        jz      Input_Error             ;yes, so there was no input included
        cmp     al,20                   ;is this a blank (space) ?
        jz      Scan_Input              ;yes, so get the next character
Store_Input:
        stosb                           ;no, this is a "real thing", save it
        lodsb                           ;check the next character
        cmp     al,Cr                   ;have we reached the end yet ?
        je      Input_Ok                ;yes, so indicate input is ok
        cmp     al,20                   ;is it a space ?
        jne     Store_Input             ;no, so keep moving characters
Input_Ok:
        clc                             ;set carry flag to zero (0)
        ret                             ;done, so return
Input_Error:
        stc                             ;set carry flag to one (1)
        ret                             ;done, so return
Get_Filename    endp

page ;*****************************************************************************
;  This routine opens the named file for reading.
;
;       INPUT   -       none
;
;       OUTPUT  -       named file is opened as read only
;                       [AX] = file handle of named file
;                       [ BP] = file handle of named file
;
;       CHANGED -       [DX]
;*****************************************************************************
Open_Input      proc near
        mov     dx,offset Filename      ;point to the input filename
        mov     ax,3d00                 ;[AL] = read only, [AH] = open file
        int     21                      ;use the DOS function call
        mov     bp,ax                   ;save the handle for later
        ret                             ;done, so return
Open_Input      endp

;*****************************************************************************
;  This routine reads the file into a buffer and places the EOF marker.
;
;       INPUT   -       [BP] has the file handle
;
;       OUTPUT  -       file is read into the buffer
;
;       CHANGED -       [AX], [BX], [CX], [DX], flags
;*****************************************************************************
Read_The_File   proc near
        mov     bx,bp                   ;get the handle of the file
        mov     cx,Block_Size           ;set [CX] to # of characters to read
        mov     dx,offset Buffer        ;point to the input file buffer
        mov     ah,3f                   ;set to read the file
        int     21                      ;use the DOS function call
        jnc     No_Read_Error           ;read was ok, so continue
        xor     ax,ax                   ;simulate a zero read on error
No_Read_Error:
        mov     bx,ax                   ;get the returned buffer pointer
        add     bx,offset Buffer        ;point to the offset into buffer
        mov     byte ptr [bx],Eof       ;place the file marker at end of input
        ret                             ;done, so return
Read_The_File   endp

page ;*****************************************************************************
;  This routine closes the read only file that was previously opened.
;
;       INPUT   -       [BP] has the file handle
;
;       OUTPUT  -       file is closed
;
;       CHANGED -       [AX], [BX]
;*****************************************************************************
Close_Input     proc near
        mov     bx,bp                   ;retreive handle of opened file
        mov     ah,3e                   ;set to close the file
        int     21                      ;use the DOS function to close file
        ret                             ;done, so return
Close_Input     endp

page ;*****************************************************************************
;  This routine displays the text file on the screen by placing it directly
;  in the display buffer memory.  It handles inbedded character attributes.
;
;       INPUT   -       [AH] = beginning display attribute (07h)
;                       [CX] = maximum size of the text file (4096)
;                       [DX] = address of display port
;                       [DI] = points to beginning of display buffer
;                       [SI] = points to beginning of text buffer
;
;       OUTPUT  -       text file is displayed on screen
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;
;       CHANGED -       [AX], [BX], [CX], [DI], [SI], flags
;*****************************************************************************
Display_Text    proc near
        lodsb                                   ;get a character into [AL]
        cmp     al,20                           ;check for imbedded control codes
        jb      Check_Character                 ;see if it is a control code
Display_Character:
        mov     bx,ax                           ;save the character/attribute in [BX]
Status_Low:
        in      al,dx                           ;get the current display status
        ror     al,01                           ;is the status low ?
        jc      Status_Low                      ;wait until it is
        cli                                     ;disable interrupts
Status_High:
        in      al,dx                           ;get the current display status
        ror     al,01                           ;is the status high ?
        jnc     Status_High                     ;wait until it is
        mov     ax,bx                           ;recover the character/attribute
        stosw                                   ;put the character in display buffer
        sti                                     ;re-enable interrupts
        loop    Display_Text                    ;loop until finished, a character/time
End_Of_File:
        ret                                     ;done so return
Check_Character:
        cmp     al,Cr                           ;is it a carriage return ?
        je      Carriage_Return                 ;yes, so handle it
        ja      Attribute                       ;it's greater so it may be an attribute
        cmp     al,Lf                           ;is it a line feed ?
        je      Line_Feed                       ;yes, so handle it
        cmp     al,Tab                          ;is it a tab ?
        je      Expand_Tab                      ;yes, so expand it
        jmp     Display_Character               ;return, because it's not an attribute
page
Attribute:
        cmp     al,Eof                          ;is it the end of the file ?
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

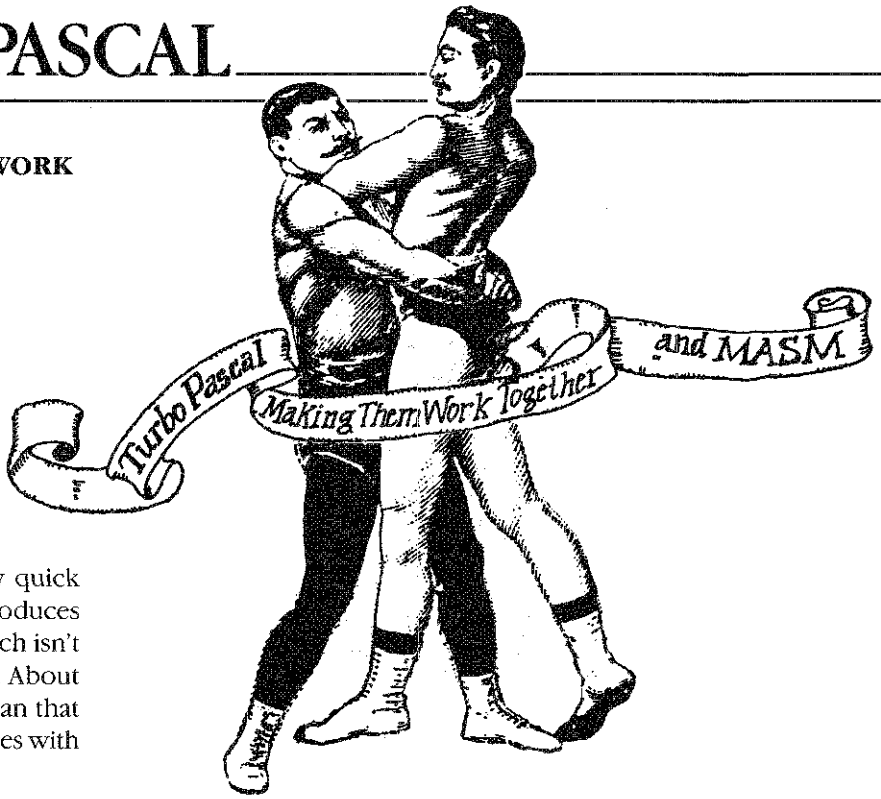

```
        je      End_Of_File             ;yes, so finish up and return from call
        jb      Is_It_High              ;no, see if high intensity
        cmp     al,Under_Line           ;should we do underlining ?
        je      Under_Line_A            ;jump if yes
        cmp     al,Blink                ;should we make it blink ?
        je      Blink_A                 ;jump if yes
        jmp     Display_Character       ;no change, so continue
Is_It_High:
        cmp     al,High_Intensity       ;should we set high-intensity ?
        je      High_A                  ;jump if yes
        jb      Is_It_Normal            ;see if a reset to normal
        cmp     al,Reverse              ;should we set a reverse attribute ?
        je      Reverse_A               ;jump if yes
        jmp     Display_Character       ;no change, so continue
Is_It_Normal:
        cmp     al,Normal               ;should we set a normal attribute ?
        je      Normal_A                ;jump if yes
        jmp     Display_Character       ;no change, so continue
Normal_A:
        mov     ah,07                   ;set display attribute to normal
        add     di,0002                 ;update display buffer pointer [DI]
        jmp     Display_Text            ;new attribute now set
High_A:
        mov     ah,0f                   ;set to high-intensity
        add     di,0002                 ;update display buffer pointer [DI]
        jmp     Display_Text            ;new attribute now set
Reverse_A:
        mov     ah,70                   ;set to reverse display
        add     di,0002                 ;update display buffer pointer [DI]
        jmp     Display_Text            ;new attribute now set
Under_Line_A:
        mov     ah,01                   ;set to underlining
        add     di,0002                 ;update display buffer pointer [DI]
        jmp     Display_Text            ;new attribute now set
Blink_A:
        mov     ah,87                   ;set to blinking
        add     di,0002                 ;update display buffer pointer [DI]
        jmp     Display_Text            ;new attribute now set
Carriage_Return:
        mov     di,Row                  ;recover address of beginning of row
        jmp     Display_Text            ;discard character and get another
Line_Feed:
        mov     bx,00a0                 ;set [BX] to number of bytes/row
        add     di,bx                   ;update [DI] the display buffer pointer
        add     Row,bx                  ;update the row count for later
          jmp     Display_Text          ;discard character and get another
page
Expand_Tab:
        shr     di,01                   ;divide current video offset by 16
        shr     di,01
        shr     di,01
        shr     di,01
        inc     di                      ;add 1 so we end with next tab position
        shl     di,01                   ;multiply by 16
        shl     di,01
        shl     di,01
        shl     di,01
        jmp     Display_Text            ;[DI] now has new tab position
Display_Text    endp                    ;discard character and get another

;------------------------------------ DATA ------------------------------------

Filename        db      40 dup (?)      ;storage for the input filename (path)
Row             dw      0000            ;storage for row count - start at 0
Port            dw      ?               ;storage for address of display card
Block_Size      dw      ?               ;maximum block size to read from file

Error_1         db 'This utility only operates while using DOS 2.0 or newer ...'
                db Cr,Lf,Lf,'$'
Error_2         db 'You must name the file to be displayed ...'
                db Cr,Lf,Lf,'$'
Error_3         db 'The Input file was not found ...'
                db Cr,Lf,Lf,'$'
page
Buffer:         db '(c) Copyright 1985, by Michael L Hoyt.'
                db '  All rights reserved.'
Code    ends
end     Display
```

□

TURBO PASCAL AND MASM: MAKING THEM WORK TOGETHER

by Mark Minasi

Mark Minasi is a partner at Moulton & Company, a professional training and consulting firm with clients in the U.S., Canada, and Western Europe. Moulton & Company train in the fields of microcomputers, artificial intelligence, and data communications. Mark Minasi can be reached at 5400 Wilson Blvd., Arlington, VA 22305, or by telephone at either (703) 276-0170 or (301) 596-6493.



Turbo Pascal is a great system for developing fairly quick routines for the PC. The compiler is very fast and produces fast code. The minimum code size is about 11K, which isn't bad compared to just about anything on the market. About the only really troublesome thing about it (other than that *awful* editor installation program) is the way it interfaces with DOS.

Borland Associates, the creators of Turbo, decided to deal with DOS on their own terms. Only in the most recent version does Turbo even recognize redirection of STDIN and STDOUT — and that's at a considerable loss in speed if you take the option. Their reason was a good one: Part of Turbo's speed comes from the fact that it doesn't wait for those glacially slow DOS service routines to get things done. (You don't think the DOS service routines are slow? Find a friend with an Apple, or a Commodore 64, or an Atari 800 and type in the one-line BASIC program 10 FOR I=1 TO 1000:PRINT "HELLO";:NEXT, then stand back! You won't be able to follow the screen. The PC won't present such a problem: While the processor is much faster, the screen handling is not.)

Turbo directly produces .COM files: It can't even produce .OBJ files. To build a program library, you must use the compiler directive $I to splice in routines from other programs. Further, because you can't use the DOS linker, linking Turbo and machine language is not straightforward. Borland has attempted to handle this problem with their own machine language interface.

There are three ways to interface Turbo with machine language:

1. EXTERNAL .com procedures
2. INLINE machine language
3. INTR or MSDOS function calls.

INLINE is really useful for short programs only. INTR/MSDOS are useful when all you want to do is get to some DOS or BIOS interrupts. But suppose you have a library of fast assembled routines that you'd like to access from Turbo? You are supposed to use the EXTERNAL procedure.

To use EXTERNAL, you assemble and link a stand-alone .COM program, say FOO.COM. You then tell Turbo about it with EXTERNAL in the procedure declaration section:

```
PROCEDURE FOO; EXTERNAL 'FOO.COM';
```

Turbo has facilities that allow you to pass parameters from and to the file. Sounds good, eh? It is, except for a major addressability problem.

The problem is that when Turbo enters FOO.COM, the code segment register CS is the same as Turbo's, and the instruction pointer IP is set to the place in Turbo's code segment where FOO.COM was incorporated. To see this better, examine a typical Turbo memory map:

```
CS:0000  Turbo library
CS:XXXX  Entry point of program
CS:YYYY  Entry point of first procedure
 ... the rest of procedures and main code ...
```

Suppose your assembler program gets loaded as the first procedure. Then on entry the instruction pointer has value XXXX. Here's where the problem comes in: When you assembled your program, the assembler started at 0000 hex, not whatever XXXX hex is. So, for example, if you coded a program that began as follows (assume the DS register has the same value as the CS register):

```
        MOV     AX,BANANA
        .... ( 50 bytes later) ...
BANANA  DW      45
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the assembler would code this as "take the value at offset 50 in the data segment, and put it in the AX register." But suppose this gets called from Turbo? Then offset 50 would be way back in the Turbo library, not at all what we intended. (The results get more interesting if you are *saving* values to a location and you accidentally modify the Turbo library in the process.) Borland tells you this when they mention in passing that "no references to the data segment must be made." This is unacceptable: Writing ASM code without a data segment is kinda tough. *Possible*, but tough.

Another problem is not exactly Turbo's: The restriction to .COM files eliminates multiple segments, intersegment jumps, and all of the rest of the restrictions on .COM files. Turbo won't let you EXTERNAL to a .EXE file.

THE SOLUTION

The addressability problem can be handled with a slightly Byzantine procedure. The procedure works well, but contains one kludge: My apologies, but I'm afraid I don't see a way around it. The general idea is the following:

1. Save the old DS
2. Determine the CS and IP
3. Compute the new DS (call it DSPRIME) such that CS:IP = DSPRIME:0000
4. Set DS = DSPRIME
5. Just to be safe, set up a local stack.

And, on exit, restore the old DS and stack. Let me stress that this handles only the data addressability problem: You're still confined to one segment for code. There *is* a way around this for the CS, but it is even more Byzantine.

How do we do steps 1 to 5? I'll take them in sequence.

Step 1: Easy. Just PUSH DS.

Step 2: The CS is immediately available: MOV AX,CS. But how to get the IP? Turbo has a function OFS(a) that returns the offset of "a," whether "a" is a variable *or* a procedure. OFS(a) is the entry value of IP. Therefore, we need only pass OFS(FOO) to FOO so that it has its IP. Turbo passes call-by-value variables on the stack. As they are pushed onto the stack in order of appearance, followed by the return address, the last one is at [bp+2]. Therefore, the offset can be read by the assembler program if the last argument is Ofs(FOO). The opening lines would be

```
mov     bp,sp               ; save stack pointer
push    ds
mov     ax,[bp+2]           ; get ip value in ax
```

Now, in order for this to work, OFS(FOO) must be the *first* parameter passed to FOO.

One more refinement: Turbo insists that you return BP, CS, DS, and SS untouched. Therefore, let's put BP in safekeeping for the moment. As SI needn't be preserved, let's use it for the moment. Rewrite the above:

```
mov     si,bp       ; save bp in a register we won't
                    ; need for a bit.  Can save to
                    ; memory later.
mov     bp,sp       ; save stack pointer
push    ds
mov     ax,[bp+2]   ; get ip value in ax
mov     bx,cs       ; get cs value in bx
```

Step 3: Calculate the DSPRIME = CS + IP/16. To do this, just shift the AX register to the right 4 times, then add to BX:

```
shr ax,1
shr ax,1
shr ax,1
shr ax,1
add ax,bx
```

Note that we could have easily done this in Turbo:

```
NewDS := CSeg + Ofs(FOO) div 16;
FOO(NewDS);
```


The Fast Path for New Users of Turbo Pascal...

For TURBO PASCAL On The IBM PC

TRAINING WHEELS™

- A comprehensive step-by-step example to help you get started.
- 30 useful & necessary procedures—ready to be included in your program—each with source code and with a simple program illustrating its use: Display help screens & menus quickly, check the DOS environment strings, determine where the program was loaded from, manipulate the archive & read-only bits, determine the ROM date & type of machine & type of display being used, retrieve and parse the command line arguments, make your program preserve the color attributes set for DOS, move a file to a different directory without copying it—and many more.
- Helps get your application going quicker and provides examples of how the DOS function calls are used from Turbo.
- 80-page user-guide plus diskette.

Available for only $25.00
(plus $2 shipping). Cal. Res. add 7%.
Telephone Orders Welcomed (408) 733-4032

LANDMARK SOFTWARE
1142 Pomegranate Court
Sunnyvale, CA 94087

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Either way, the result is the same. But what if Ofs(FOO) is not on a paragraph boundary, i.e., not evenly divisible by 16? Which leads us to . . .

Step 3A: How can you ensure that Turbo will place your procedure on a paragraph boundary? In MASM, it's several simple pseudo-op instructions. In Turbo, however, no such command exists. Here we must (sigh!) result to an inelegant, if dependable, solution.

Here is how it works: The *first* procedure defined in the code is always located directly after the libraries. Therefore, its offset will be constant, depending on the version of Turbo you are using (2.0, 3.0, 8087, BCD, etc.). By putting a dummy procedure first, you can control whether or not your *second* procedure is located on a paragraph boundary. The first procedure, DUMMMY, should be a simple WRITELN with a variable length string. You never call the procedure: It simply moves code around enough to get your code on a paragraph boundary. In the case of Turbo 3.0 non-8087, you use the following:

THE HAMMER

Software Tools in C

More than just BIOS/DOS access, THE HAMMER Library also provides routines for multi-level 123-like menus, easy data entry & verification of strings, numbers, & dates, screen attribute control, date & time processing, AND MORE.

Super data entry routines give programmer a natural, strong interface with the user. They work in both "single-field" and "multi-field" (full screen editing) modes. This is NOT just another library of general purpose routines.

For: IBM/PC family with DOS 2.00+
C Compilers: CI-C86, Lattice, DeSmet, Mark Williams and new Microsoft V3.00.

$195 with source code and manual. To order or inquire, CALL OR WRITE:

O.E.S. SYSTEMS
1906 Brushcliff Road
Pittsburgh, PA 15221
412/243-7365

Looking for the right tool for the job?
GET THE HAMMER


```
Program Whatever; <--- You can change this

                  <--- Put your LABEL, CONST, TYPE, VAR sections
                       here

Procedure DUMMMY; <--- Do this without change
BEGIN
  WRITELN('01234567890123') <--- The length of string is the key
END;

Procedure FOO(A,B,...,Ofs(FOO));  <--- The name of your .COM
program is

 ... the rest of the program...
```

The length of the string in the WRITELN is what controls the alignment of the procedure FOO. The following table summarizes the values for the versions that I had on hand:

Version	8087?	Writeln string
1.0	no	''
2.0	no	'1234567'
2.0	yes	'12'
3.0	no	'123456789012'
3.0	yes	'123456789'

Note that I do not have the BCD version of 3.0, and no 8087 version exists for version 1.0. These values are not changed by re-installing Turbo, device drivers, logical devices, or the like. They are solely version-dependent.

If new versions come out, or you don't trust me, you can find the right values for DUMMMY with the following Turbo program:

```
program sizeoffset;

var r:integer;

procedure dummmy;
begin
writeln('1234567890123456');
end;

procedure second;
begin
end;

begin
  r := ofs(second) - (ofs(second) div 16) * 16;
  writeln('Make the writeln string ',r,' characters
  shorter.');
end.
```

Run this program. It will instruct you how to shorten the writeln line in DUMMMY. Then save *just* Procedure DUMMMY in a file called DUMMMY.PAS and include it in your program. By the way, this works as well for memory compilation as it does for .COM file compilation.

Again, it's a kludge, but I can't find a better one.

Steps 4 and 5: Set the new DS and set up a local stack. The assembler intro now looks like:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

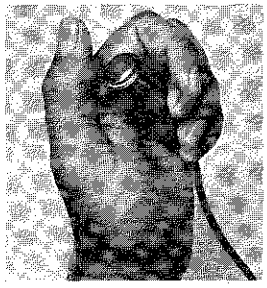
```
jmp         start               ; jump over data.  This will work,
                                ; as the jump is a 16 bit offset.

stackarea   db                  50 dup ('stack')
stacktop    label               near
savess      dw                  ?
savesp      dw                  ?
savebp      dw                  ?
start label near
      mov   si,bp               ; save bp in a register we won't
                                ; need for a bit.  Can save to
                                ; memory later.
      mov   bp,sp               ; save stack pointer
      push  ds
      mov   ax,[bp+2]           ; get ip value in ax
                                ; Note that the parms start at bp+2
                                ; because bp points to return addr.
      mov   bx,cs               ; get cs value in bx
;
; divide IP, now in AX, by 16:
;
        shr ax,1
        shr ax,1
        shr ax,1
        shr ax,1
        add ax,bx
;
; Load new DS
;
        mov ds,ax
;
; Set up local stack, save old bp
;
        cli
        mov ax,ss
        mov savess,ax
        mov savesp,sp
        mov ax,ds
        mov ss,ax
        mov sp,offset stacktop
        mov savebp,si
        sti
;
;       If there are any other parameters to retrieve, do it here
;

;
; ---> Insert your program here <---
;


; Restore stack and ds
        cli
        mov bp,savebp
        mov ax,savess
        mov ss,ax
        mov sp,savesp
        sti
;
; Return any parameter values here
;
;
; Restore DS
;
        pop ds
        ret nparms <-- put # of parameter bytes here: see Turbo
        manual
```

To make this useful, save the top and bottom parts in an assembler macro library and call the macros in your assembler program.

A NOTE ON THE RETURN

The manual doesn't say it, but in order for the procedure to return correctly, it must be coded as a PROC NEAR. This way, an intrasegment return will be coded. If you use PROC FAR to start the program, the assembler will code an intersegment return.

Introducing Periscope II Professional Debugger and Break-out Switch



New Periscope II includes a remote break-out switch that does not need its own slot!

"Periscope is: "the best value in development tools currently on the market . . . the most essential element of my developer's toolbox". —Jeff Garbers

"Very powerful for debugging and testing . . . Better than Atron by far." —Wynn Bailey

The break-out switch ***"really sets Periscope apart from the typical software-only debuggers."*** Hung system or locked keyboard? Press the switch to get control!

Periscope's symbol support ***"beats the daylights out of snooping through a map file and making notes"***. See high-level line numbers and source code, too!

"Feel right at home" **in no time** with commands that logically extend Debug's!

Periscope's speed makes other debuggers ***"look absolutely sluggish"***! It's written entirely in assembler and uses DOS only for file access.

Has all the standard features plus:

- Debug with over 75 breakpoint options
- New! Write your own breakpoint tests
- New! Traceback
- New! Do in-line symbolic assembly
- Debug using one or two monitors
- Recall command lines
- New! Debug with high-level source code
- New! Redefine windows while debugging
- New! View text files while debugging
- Debug device drivers, non-DOS and memory-resident programs
- New! Customize Periscope via user exits
- New! Display 8087/80287 status
- New! Use Periscope with an EGA

Periscope requires: IBM PC, XT, AT, or close compatible; DOS 2.0 & later; 128K RAM; 1 Disk Drive; 80-column Monitor.

Periscope II, break-out switch, manual, reference card and software . . . $145!

Periscope I also includes the write-protected RAM board to protect crucial debugger code. It's just $295!

The US Navy gets Periscopes from us . . . shouldn't you? Order today!

Order/Information Call Toll-Free:

800-722-7006

30-Day Money-Back Guarantee

Data Base Decisions • 404/256-3860
14 Bonnie Lane • Atlanta, GA 30328

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The return must specify how many bytes to pop off the stack before using stack information to develop a return address. The Turbo manual is a little sketchy on this, but it is 4 bytes for each call-by-name variable and varies for call-by-value variables:

Type	# Bytes
Integer	2
Real	6 for non 8087, 8 for 8087 version
String	String length + 1
Byte	2
Char	2
Boolean	2
Set	32
Pointer	4

So if you had the following procedure calls, you would have the corresponding number of bytes to pop off before return:

```
Procedure X(A,B,C:INTEGER);    needs   RET 6
Procedure X(A,B,C:INTEGER;VAR Y:REAL);   needs   RET 10
Procedure X(A,B,C:INTEGER;Y:REAL);   needs   RET 12 if non 8087
   and 14 if 8087
Procedure X(A,B,C:INTEGER;D:STRING20);   needs   RET 27
Procedure X(A,B,C:INTEGER;VAR D:STRING20);   needs   RET 10
```

AN EXAMPLE

I hope that this has been easy to follow so far. Personally, however, I seem to be unable to understand anything without an example. So, let's contrive one. Say I want a routine SYSMSG that takes an integer argument. The integer then tells the routine which of a list of messages to print. The messages will be as follows:

Message Number	Message
1	"Fatal error"
2	"Severe error"
3	"Warning"
4	"System Ready"

So, for instance, the code

```
BEGIN
  SYSMSG(2,Ofs(SYSMSG));
END.
```

would put "Severe error" on the screen. (I told you it was contrived.) To display a string on the screen, just point DX to the offset of the beginning of the message, then terminate the message with a dollar sign. We end up with the following code:

```
clear   segment
        assume cs:clear,ds:clear
progl   proc near
        jmp         start    ; jump over data.  This will
                             ; work, as the jump is a 16 bit
                             ; offset
stackarea           db       50 dup ('stack')
stacktop            label    near
savess              dw       ?
savesp              dw       ?
savebp              dw       ?
message1            db 13,10,'Fatal Error',13,10,'$'
message2            db 13,10,'Severe Error',13,10,'$'
message3            db 13,10,'Warning',13,10,'$'
message4            db 13,10,'System Ready',13,10,'$'
othermessage        db 13,10,'Illegal value ('
invalue             db 'X) passed to SYSMSG',10,13,'$'
start   label       near
        mov         si,bp          ; save bp in a register we won't
                                   ; need for a bit.  Can save to
                                   ; memory later.
        mov         bp,sp          ; save stack pointer
        push        ds
        mov         ax,[bp+2]      ; get ip value in ax
        mov         bx,cs          ; get cs value in bx for
                                   ; calculating ds
;
; divide IP, now in AX, by 16:
;
        shr ax,1
        shr ax,1
        shr ax,1
        shr ax,1
        add ax,bx
;
; Now it's ready: load new DS
;
        mov ds,ax
;
; Set up local stack, save old bp
;
        cli
        mov ax,ss
        mov savess,ax
        mov savesp,sp
        mov ax,ds
        mov ss,ax
        mov sp,offset stacktop
        mov savebp,si
        sti
;
; *************** END OF INTRO CODE **********************
; *************** START USER CODE ************************
;
; First, get second parameter, an integer.
;
        mov ax,savess
        mov es,ax
        mov ax,es:[bp+4]  ; second parameter
;
; Load offset of message 1, 2, 3, or 4 depending on value of AX.
;
        cmp ax,1
        jne ck2
        mov dx, offset message1
        jmp printit
ck2:    cmp ax,2
        jne ck3
        mov dx, offset message2
        jmp printit
ck3:    cmp ax,3
        jne ck4
        mov dx, offset message3
        jmp printit
ck4:    cmp ax,4
        jne ck5
        mov dx, offset message4
        jmp printit
;
; Illegal message: only reproduce lower byte of ax
;
ck5:    add ax,30h
        mov invalue ,al
        mov dx,offset othermessage
;
; Print message
;
printit:
        mov ah,9
        int 21h

; ************* END OF USER PROGRAM
; ************* RETURN-TO-TURBO CODE BEGINS...
;
; Restore stack
        cli
        mov bp,savebp
        mov ax,savess
        mov ss,ax
        mov sp,savesp
        sti
;
;
; Restore DS
;
        pop ds
        ret 4 ; 4 bytes because 2 for Ofs(), 2 for message switch
progl   endp
clear   ends
        end progl
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Save this into a file named ATEX.ASM ("Assembler to Turbo EXample") and assemble, link, and convert to a .COM file. Then create the following ATEX.PAS program in Turbo:

```
Program TestMsg;

(* Demonstrates linking a separate stand-alone assembler file with
   Turbo.  *)

Procedure Dummmy;
BEGIN
writeln('123456789012')

(* This is for turbo version 3.0 without 8087.  See article text
for mods for other versions *)

END;

Procedure Sysmsg(msgnum,offst:integer) ;EXTERNAL 'B:ATEX';

BEGIN (* Main program *)

  Sysmsg(1,Ofs(Sysmsg)); (* Will produce 'Fatal Error' message *)
  Writeln('Returned from Sysmsg');
  Sysmsg(2,Ofs(Sysmsg)); (* Will produce 'Severe Error' message *)
  Writeln('Returned from Sysmsg');
  Sysmsg(8,Ofs(Sysmsg)); (* Will produce 'Illegal Value' message *)
   Writeln('Returned from Sysmsg');

END.   (* Main program *)
```

CASES WHERE DS ISN'T NECESSARY

The preceding discussion is relevant only if you absolutely must have control of the DS register. For most any but the smallest programs, this is a must. There are, however, situations where it may not be necessary. I will discuss creating these assembler programs in the remainder of this article.

Obviously, steps 1-5 above are no longer necessary. What are the programmer's responsibilities if he/she does not need the DS register?

1. Must save BP, SS, DS, and CS
2. Must pop (as before the correct number of bytes off with RET
3. RET must be (again) intrasegment.
4. The offset need not be passed to the assembler routine.

As an example, I have written a small program, called ATEX2.ASM, that accepts three parameters A, B, and C. A and B are called by value, C by name. The values of A and B are added, and their sum is returned through C. Recall from above that this means that the stack requirements are 2 + 2 + 4 = 8 bytes.

The program is simple. First, it saves bp on the stack. Then it retrieves the values of A and B. Since it is called with parameters in order:

```
(A,B:integer; VAR C:Integer)
```

the old value of BP is first on the stack (we just put it there), the return address is next on the stack, then the 4 bytes for C's address, then the 2 bytes for B's value, and finally the 2 bytes for A's value. Thus, [SP] points to BP's old value, [SP + 2] to the return address, [SP + 4] to C's address, [SP + 8] to B's value, and [SP + 10] to A's value. Hence, we can easily retrieve and sum the values of A and B with the following lines of code:

```
mov ax,[bp+8]
add ax,[bp+10]
```

There remains only the problem of saving C. The four bytes are pushed onto the stack with segment first, followed by offset. Thus, as the value to be returned is in AX,

```
push es                 ; save es before destroying
mov bx,[bp+6]           ; get the segment
mov es,bx               ; load into es
mov bx,[bp+4]           ; now get the offset
mov es:[bx],ax          ; and save the value
```

The entire code follows:

```
;
;
; This is a sample assembly function for Turbo Pascal.  It takes
; two integers as inputs, then adds them and places the result in
; the third.

clear   segment
saver   dw
        assume cs:clear
progl   proc near
        push bp
        mov bp,sp
        ;
        ; Get the 1st and 2nd parameters.
        ;
        mov ax,[bp+8]          ; Get B
        add ax,[bp+10]         ; Add to A
        ;
        ; Now save back to original location of last variable, C
        ;
        push es                ; save es before destroying
        mov bx,[bp+6]          ; get the segment
        mov es,bx              ; load into es
        mov bx,[bp+4]          ; now get the offset
        mov es:[bx],ax         ; and save the value
        ;
        ; Now restore the state of the registers on entry
        ;
        pop es
        mov sp,bp
        pop bp
        ret  8
progl   endp
clear   ends
        end progl
```

Save this as ATEX2.ASM. Assemble, link, and convert to a .COM file. The corresponding Turbo program follows.

```
program testasm;

var a,b,c:integer;

procedure addbc(a,b:integer;var c:integer); external 'ATEX2';

begin   (* Main program *)
  a:=320;
  b:=7;
  addbc(a,b,c);
  writeln(a,' ',b,' ',c);
end.    (* Main program *)
```

CONCLUSION

I have discussed linking Turbo Pascal to assembled .COM programs, discussed drawbacks in the current Turbo/MASM link, and offered a dependable (if somewhat kludgy) solution. I am a little uncomfortable with the method used to ensure that the .COM file is loaded on a paragraph boundary, more from the standpoint of elegance than anything else. Does anyone know another way? Please let me know.

Until then, I hope that you find this useful. Turbo is great — gets better each release — but some things just need assembler for speed or system function access. My routines demonstrate how to make them work together. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

dBASE II ROUTINES

by Brett Salter

Brett Salter of Data Base Decisions sent us these dBASE II routines. ANSI prints a string to the monitor. COMPRESS squeezes a string into a byte. EXPAND turns the bits of a byte into ASCII ones and zeroes. LPTSWAP swaps parallel printers 1 and 2. IN returns the value of a port. OUT changes the value of a port. And DBFIX rounds the size of a dBASE II file up to the next even multiple of bytes to avoid possible file truncation.

```
csegsegment para public 'code'
org100h

; ansi
; print string to monitor

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0dfffh; sp value

dsubproc near

mov cl,[bx]; length of string
mov ch,0
inc bx; point to first char

p030:mov dl,[bx]; get char
mov ah,2
int 21h ; display char in dl
inc bx
loop p030; done?

p100:ret; return to dBASE

dsubendp
csegends
enddsub



csegsegment para public 'code'
org100h

; compress
; compress string into a byte

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0dfffh; sp value

dsubproc near
jmp short p010; start point

a000equ this word
bytedb 0; result

p010:call p020; get address
a020equ this byte
p020:pop si
sub si,a020-a000; get address of data
mov al,0
mov [si],al; clear output

mov cl,[bx]; get length of string
cmp cl,8; more than 8?
ja p100 ; yes
mov ch,0

inc bx; point to input
mov al,0

p025:mov ah,[bx]; get byte
cmp ah,'0'
jb p100 ; too low
cmp ah,'1'
ja p100 ; too high
sub ah,'0'                ; make it binary

shl al,1
add al,ah; add in new bit

inc bx; point to next
loop p025; done?

mov [si],al; save the byte

p100:ret; return to dBASE

dsubendp
csegends
enddsub



csegsegment para public 'code'
org100h

; expand
; expand bits of a byte into ascii ones & zeroes

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0dfffh; sp value

dsubproc near
jmp short p010; start point

a000equ this word
bytedb 0; input byte

p010:call p020; get address
a020equ this byte
p020:pop si
sub si,a020-a000; get address of data

mov al,[bx]; check length of string
cmp al,8; 8?
jnz p100; no -return

mov al,[si]; get input byte
inc bx; point to output

mov cx,8
mov ah,80h; compare pattern
mov dx,'01'

p025:test al,ah
jz p030 ; bit is off

mov [bx],dl; bit is on
jmp p040

p030:mov [bx],dh
p040:inc bx
shr ah,1
loop p025; done?

p100:ret; return to dBASE

dsubendp
csegends
enddsub



csegsegment para public 'code'
org100h

; lptswap
; swap parallel printers 1 & 2

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0dfffh; sp value
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
dsubproc near
push es
xor ax,ax
mov es,ax
mov di,408h
mov ax,es:[di]; printer 1
mov bx,es:[di+2]; printer 2
mov es:[di],bx
mov es:[di+2],ax
pop es
p100:ret; return to dBASE

dsubendp
csegends
enddsub
```

```
csegsegment para public 'code'
org100h

; in
; return value of port

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0df00h; sp value

dsubproc near
jmp short p010; start point

datumequ this word
portdw 0; port #
valdb 0; value

p010:call p020; get address
offerequ this byte
p020:pop si
sub si,offer-datum; get address of data
mov dx,[si]; get port no
in al,dx; read the port
mov [si+2],al; save it
p100:ret; return to dBASE

dsubendp
csegends
enddsub
```

```
csegsegment para public 'code'
org100h

; out
; change value of port

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0df00h; sp value

dsubproc near
jmp short p010; start point
datumequ this word
portdw 0; port #
valdb 0; value

p010:call p020; get address
offerequ this byte
p020:pop si
sub si,offer-datum; get address of data
mov dx,[si]; get port no
mov al,[si+2]; value to out
out dx,al; write the port
p100:ret; return to dBASE

dsubendp
csegends
enddsub
```

```
csegsegment para public 'code'
org100h
dbfixproc far
```

```
; This program is used to round the size of a dBASE II file up to the next
; even multiple of 128 bytes, to avoid possible file truncation. The program
; is called using 'DBFIX filename' from DOS, where filename is the name of
; the file.


  assume cs:cseg,ds:cseg,ss:nothing,es:nothing

  p010:mov dx,offset copyr; display copyright
  call p200
  mov dx,5ch; point to first fcb
  mov ah,0fh
  int 21h ; open file
  cmp al,0; error?
  jz p020 ; no
  mov dx,offset info; tell the user about us
  call p200
  int 20h ; and terminate

  p020:mov si,5ch
  mov ax,[si+16]; get low part of file size
  and ax,7fh; any remainder?
  mov dx,offset nomod; no modification
  jz p080 ; no
  mov remainder,ax; save remainder
  mov ax,[si+16]; get low part again
  mov cl,7
  shr ax,cl; divide by 128
  mov bx,[si+18]; modify high word
  mov cl,9
  shl bx,cl
  jc p080 ; overflow
  add ax,bx; no of 128 byte recs
  mov [si+33],ax
  mov al,0
  mov [si+35],al

  mov ah,27h; get record
  mov cx,1
  mov dx,5ch
  int 21h


  mov ax,[si+33]; modify file size
  dec ax
  mov [si+33],ax
  mov di,remainder
  add di,128; point to start of expansion
  mov cx,256
  sub cx,di; get bytes to zap
  mov al,0
  rep stosb

  mov ah,28h
  mov cx,1
  mov dx,5ch
  int 21h

  mov ah,10h; close the file
  mov dx,5ch
  int 21h
  mov dx,offset modify

p080:call p200
int 20h

p200proc near; display message
push ax
mov ah,9
int 21h
pop ax
ret
p200endp

copyrdb 'DBFIX - Copyright 1983 Data Base Decisions',10,13,'$'
infodb 'This program is used to round the size of a dBASE II file up to'
db 10,13
db 'the next even multiple of 128 bytes, to avoid possible file trun-'
db 10,13
db 'cation. To use this program, enter "DBFIX filename", where file-'
db 10,13
db 'name is the file to be fixed.',10,13,'$'
nomoddb 'File not resized',10,13,'$'
modifydb 'File resized',10,13,'$'
remainder dw 0

dbfixendp
csegends
enddbfix
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TERMINAL EMULATION $49

With XTERM your PC can emulate the DEC VT100/102, VT52, IBM3101, HP2624, ADDS, and ADM3A. Site licenses available for all products.

BASICA WINDOWS $49

Enhance your basica programs with a dozen new window commands for both interpreter or compiler. Add $399 for complete "ASM" source code.

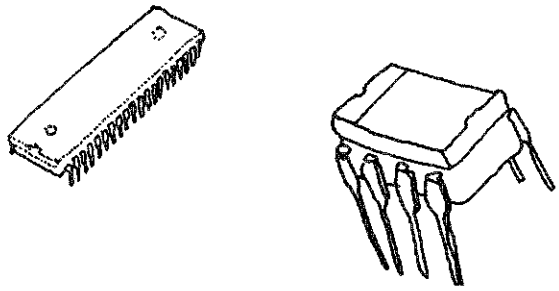
DES ENCRYPTION $59

Government approved data encryption standard with compression and telecommunications formatting. Add $399 for complete "C" source code.

REMOTE OPERATIONS $69

Operate your PC/DOS remotely via telephone using CONSOLX. Great for LAN or dBase II applications. Add $399 for complete "ASM" source code.

HAWKEYE GRAFIX
23914 Mobile Street
Canoga Park, CA 91307
(818) 716-5220



GREAT MOMENTS IN PERSONAL COMPUTER HISTORY:
An unknown programmer senses that for the first time in hardware history, the chip is about to hit the fan.

TallScreen™ TAKES THE DRUDGERY OUT OF DOS

- 1000 lines of screen output saved to scroll through for review and editing
- Commands and directory paths automatically saved for quick recall again and again
- Screen output and commands edited with standard, powerful, easy-to-learn and easy-to-use word processor techniques
- Screen output saved to file or printer, or frozen anywhere on the screen
- DOS commands help file included for quick reference
- Keyboard macros supported similar to the IBM Personal Editor.

For the IBM PC®, PCjr, XT, AT, or compatibles. Works with DOS 2.00 and above. Works with PROKEY™ and SIDEKICK™. Compatible with most mouse systems. Next day shipping on VISA/MC. MD & NM residents add sales tax.

$49.95
INTRODUCTORY PRICE

"TallScreen has completely changed the way I use DOS."
Bob Beilstein, PC user

"A godsend for power users."
Hanno Hinsch, Author of TELIOS

"This is the way DOS should have been designed in the first place!"
Rex Swain, Technical Director Parallax Systems

"Just recalling screen output and past commands alone is worth the price."
Edward T. Garner, VP Finance ARIES Group

For orders or information see your local dealer or call 800-522-0290

QUALITAS, INC., P.O. BOX 3AK/UPB
LAS CRUCES, NM 88003 505-522-0290

QUALITY PROGRAMMING TOOLS
QUALITAS™

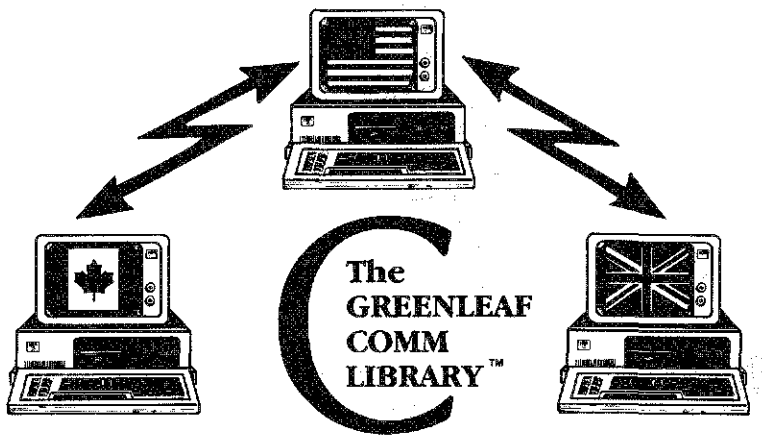
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ARE YOU TRYING TO COMMUNICATE ?

C programs can communicate with the world now through the power of **The Greenleaf Comm Library.** Now from the people who brought you **The Greenleaf Functions** General Library for C, comes this rich interrupt driven, ring-buffered asynchronous communications capability.

Over 100 functions in C and assembler to facilitate communications at up to 9600 baud. Up to eight ports at a time. ASCII or XMODEM. X-On/X-Off too. Hayes compatible modems controlled here. Safe too, bet you can't exit your application with interrupts hot. Major applications around the world base their communicating applications on **The Greenleaf Comm Library.** Stop just trying and start really communicating. Get your copy of **The Greenleaf Comm Library** today. For all major C compilers, all models, all versions. For the IBM PC and just about any machine with MSDOS and an 8086. Comes with source code, extensive examples, demo programs, featuring **C-Terminal,** reference card and newsletter. No royalty. $185

Other Products: The Greenleaf Functions General Library, over 220 functions for total control of the IBM PC, with source. $185 for the compilers listed below. (See ordering instructions below).



Specify compiler when ordering: Lattice, Microsoft, Computer Innovations, Mark Williams, or DeSmet. Add $7.00 for UPS Second Day Air (or $5.00 for ground). Texas residents add sales tax. Mastercard, VISA, check, or P.O. In stock, shipped same day.

☐ General Libraries	$185
☐ Comm Library	$185
☐ CI186 Compiler	$349
☐ Lattice C	$395
☐ Mark Williams	$475

PRICES ARE SUBJECT TO CHANGE WITHOUT NOTICE.

For Information:
214/446-8641

GREENLEAF SOFTWARE©

2101 HICKORY DR.
CARROLLTON, TX 75006

1	2
3	4

Have we got your number?

Is Programmer's Journal reaching you at the correct address? Please notify us of any address change or correction.

Is it nearing time to renew? Your Programmer's Journal mailing label shows when your subscription expires. Renew early to avoid any interruption in your subscription service!

Programmer's Journal, P.O. Box 3000, Dept. EE, Denville, NJ 07834

GROWING OLD?

...waiting for C programs to compile and link?

Use C-terp
the complete C interpreter

This is the product you've been waiting (and waiting) for!

Increase your productivity and avoid agonizing waits. Get instant feedback of your C programs for debugging and rapid prototyping. Then use your compiler for what it does best...compiling efficient code ...slowly.

C-terp Features

- Full K&R C (no compromises)
- Complete built-in screen editor-- no half-way house, this editor has everything you need such as multi-files, inter-file move and copy, etc. etc. For the ultimate in customization, editor source is available for a slight additional charge of $98.00.
- Fast--Linking and semi-compilation are breath-takingly fast. (From edit to run completion in a fraction of a second for small programs.)
- Convenient--Compiling and running are only a key-stroke or two away. Errors direct you back to the editor with the cursor set to the trouble spot.
- Object Module Support-- Access functions and externals in object modules produced by your compiler. **New:** We are now supporting **Microsoft** 3.0, **Mark Williams** & **Aztec C** in addition to C.I. C86 & Lattice.
- Complete Multiple Module Support.
- Symbolic Debugging-- Set breakpoints, single-step, and directly execute C expressions.
- Many more features including batch mode and 8087 support.
- **Price: $300.00 (Demo $45.00) MC, VISA**

Price of demo includes documentation & shipping within U.S. PA residents add 6% sales tax. Specify compiler.

- C-terp runs on the IBM PC (or compatible) under DOS 2.x with a suggested minimum of 256Kb of memory. It can use all the memory available.

GIMPEL SOFTWARE

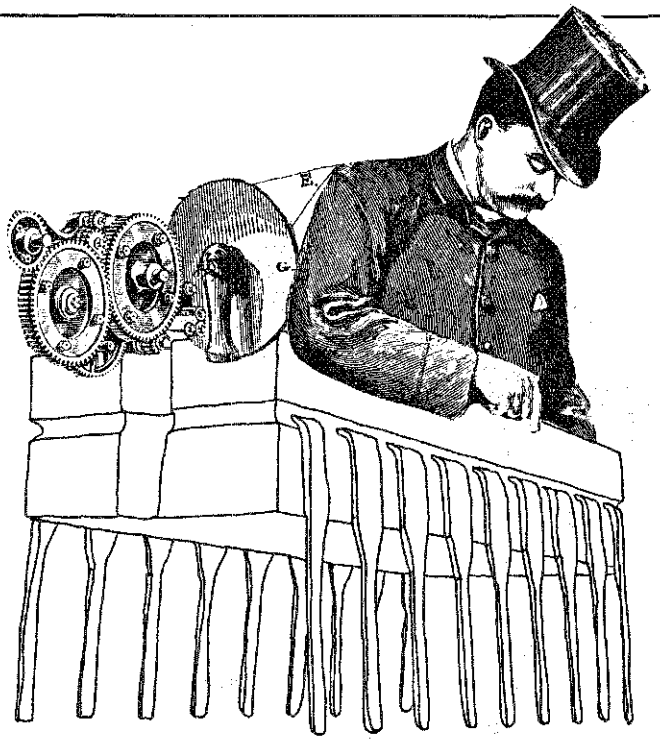
3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

*Trademarks: C86 *(Computer Innovations)*, Lattice *(Lattice Inc.)* IBM *(IBM Corp.)*, C-terp *(Gimpel Software)*, Microsoft *(Microsoft)*, Aztec *(Manx)*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

GETTING USED TO TURBO PASCAL 3.0

by Tom Swan



Turbo Pascal breaks all the rules. It's fast, reliable, and inexpensive. It comes with a lightning fast editor and a good reference manual, the disk is not copy protected, and there are no royalties on selling compiled code. And, contrary to established norms, Borland International keeps coming out with these updates — adding features, fixing bugs, and increasing the speed of their system. For heaven's sake, they even offer free telephone support.

My question is, when is Borland going to start playing by the rules, painstakingly developed over the years by compiler writers? We all know you have to pay at least $500 for a real compiler. We expect to write our way around bugs. We want to pay a royalty of twice the retail price of our products. We like to call and listen on hold to Willie Nelson recordings for 20 minutes. And please, don't fix it — even if it is broken. We won't be able to understand the "Bugs Fixed" list anyway.

It's obvious, with Borland's newest release of Turbo Pascal Version 3.0, that Philippe Kahn is just never going to listen to reason. I received my update in two weeks, before I even had time to start worrying about my order's being lost. I looked and looked, but no, there were no mimeographed update sheets anywhere to be found — just a brand new manual, which I suppose will have to do. Resigned to Borland's way, I'm ready to put my old compilers up on the shelf — preferably next to the teletype where they'll be real handy. Now that I'm stuck with it, here are a few things I'm getting used to in Turbo Pascal 3.0.

PARAMETERS

You can pass parameters to programs. This feature lets you write quiet programs — those with no prompts or other messages. For example, you could write a program SORT, and then from the DOS command prompt, enter this:

```
A>SORT SOURCE.TEXT DEST.TEXT
```

Presumably, SORT would then operate on the two files, SOURCE.TEXT and DEST.TEXT. Two new functions give access to the parameters. Function ParamCount returns the number of parameters passed to the program. Function ParamStr(N) returns the text of parameter N in the form of a string. The following program demonstrates the two new functions:

```
PROGRAM Parame;
VAR
   n : integer;
BEGIN
   writeln( 'Number of parameters: ', ParamCount );
   FOR n := 1 TO ParamCount DO
      writeln( n:2, ' : ', ParamStr(n) )
END.
```

Your program can also check ParamCount, and if it's less than the required number, prompt for the missing parameters, or print instructions about what parameters are required.

If you don't compile your program to a .COM file, a new compiler option simulates parameter passing. Press O at the main command page, and then select option P. Enter the parameters separated by blanks and then go back to the command page. When you run the program in memory, it will use the parameters you entered. This feature lets you debug a parameterized program without having to compile it to a .COM file.

I/O REDIRECTION

Two new compiler options, G and P, define input (G) and output (P) buffer sizes for redirecting I/O. Each of these options takes an integer value equal to the byte size of the buffer you want to use. For example, {$G512,P512,D-} allocates input and output buffers of 512 bytes each.

The D- option, also new, tells Turbo Pascal to turn off device checking for Reset, Rewrite, or Append operations. By doing this, Turbo allows normally disallowed buffering of character devices.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

With buffered character devices, the standard Pascal files Input and Output go through DOS file handles. These are redirected by specifying input (<) and output (>) files in the DOS command line. The following program demonstrates these new features.

```
{$G512,P512,D-}
PROGRAM Upper;
{Convert text to upper case}
VAR
   ch : char;
BEGIN
   WHILE NOT Eof DO
   BEGIN
      read( ch );
      write( UpCase(ch) )
   END
END.
```

The program converts text to upper case. Save it as UPPER.PAS and compile to a .COM file. You can make the program turn itself into upper case, sending output to the console, with this command:

```
A>UPPER <UPPER.PAS >CON:
```

The text of UPPER.PAS is given as input to program UPPER, which directs its output to file CON:. Notice the use of Eof with no parameter. The manual says (on page 361) that to do this requires input buffering to be turned off with the compiler option {$B-}. Methinks this is misleading because turning off device checks with {$D-} turns on input buffering, just the opposite of what the manual suggests.

SUB-DIRECTORIES

All levels of Turbo Pascal, including the editor, now accept full DOS path names. Previously, it was difficult to keep a library of procedures in one directory and the main source programs in another, as you would probably want to do on a hard disk. In earlier versions, to include a file from a library, you had to either transfer the file to your working directory, or manually change directories, load the routine, then go back to wherever you came from.

Now, after loading a file from a sub-directory, if you change directories, Turbo Pascal remembers where the original file was located. If you make any changes to the file, it will be stored back in its original sub-directory. To store the updated file elsewhere, you now have to mark it with function keys F7 (begin block) and F8 (end block), then use the command ^KW to write the marked text to a named file.

Three new procedures, ChDir, MkDir, and RmDir, let you change, make, and remove sub-directories from inside programs. You may, of course, now assign full path names to your own program files for resetting and rewriting.

EDITOR CHANGES

The editor is now super fast, and Borland has done away with the annoying cursor "noise" that I used to see while paging on my monochrome display. I suspect the new version writes directly to video memory and avoids sending the cursor to each character position. It's hard to say exactly, but the editor seems to page about four times faster than before. The 2.0 editor was slower than WordStar; now it's faster.

NEW SIZES

Editor work space decreased 2,083 bytes down to 60,552 bytes from 62,635 in version 2.0. You can always break large programs into pieces, however, and compile them with a dummy shell that includes all the parts, so this is no great loss.

The size of the TURBO.COM file itself increased 3,191 bytes, up to 39,671 bytes from version 2.0's 36,480. Unless my calculator is broken, that means the new system uses memory more efficiently by 1,108 bytes (3,191 – 2,083) as the increase in compiler size is that much greater than the decrease in editor work space.

A program's compiled .COM file size increased 1,381 bytes. Compiling a null, no-statement .COM program occupied 10,052 bytes under version 2.0. The same program now occupies 11,433 bytes in version 3.0.

NEW SPEED

To test Borland's outrageous speed claims (which turned out to be not so outrageous after all), I ran three tests.

First, I executed a dice simulation, throwing 10,000 times. I used a stopwatch to measure speed, so these results are probably no more accurate than plus or minus one second. Next, I ran the standard Byte Sieve benchmark. After that, I compiled to a .COM file on a PC XT hard disk my 3,547-line editor program from my book, *Pascal Programs for Data Base Management*. Following are the results of these tests.

Test	2.0 time	3.0 time	Remarks
DICE PROGRAM	00:27	00:16	(10,000 iterations)
SIEVE	00:17	00:15	(1,899 primes)
EDITOR	01:59	01:12	(compile to .COM file)

My comparisons tell me that Turbo Pascal 3.0 compiles 40 percent faster and runs anywhere from 12 to 60 percent faster than version 2.0. The Sieve benchmark was disappointing, but the other times indicate that Borland's advertised claims of twice the speed are not to be discounted.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

RUN TIME DIFFERENCES

In all previous versions of IBM PC Turbo Pascal, a program automatically cleared the screen when it began. Countless numbers of programmers (at least I didn't count them), sent patches to CompuServe to defeat this feature, which never existed in CP/M versions of the compiler.

In Turbo Pascal 3.0, if you want the screen to clear, you have to execute ClrScr. Otherwise, there's no change to the display or cursor position when a program begins running.

GRAPHICS

There is a whole array of new graphics procedures stored separately in two files, GRAPH.P (containing text headers) and GRAPH.BIN (containing machine language external procedures). To use the new routines, a program includes GRAPH.P. For example, the following program draws a circle and fills it with RED.

```
PROGRAM RedEye;
CONST
   x = 160;
   y = 100;
{$I GRAPH.P}
BEGIN
   GraphColorMode;
   Circle( x, y, 50, 1 );
   FillShape( x, y, 2, 1 );
   readln;
   TextMode
END.
```

Actually, there are two levels of PC graphics available. On the first level, you get a few basic screen control procedures such as TextMode, TextColor, GraphColorMode, GraphMode, and HiRes, which turn on various screen modes and colors. You can also set the color Palette, draw lines, plot points, and restrict graphics to a limited window.

On the second level, which you achieve by including the GRAPH.P file, you get additional procedures such as Arc, Circle, FillScreen, FillShape, and FillPattern. GetPic and PutPic are similar to the BASIC commands GET and PUT. These two procedures copy bit-mapped shapes into buffer variables or write them on the screen.

Also included in GRAPH.P is a set of Turtle Graphics routines. With these procedures, you manipulate an imaginary turtle (whose body you can make visible if you want). By leaving a trail as the turtle moves, your program draws lines and shapes. You can set the turtle's Heading, move it Forwd and Back, put PenDown or PenUp to start or stop drawing in a color selected by SetPenColor, and you can TurnLeft, TurnRight, and send the turtle Home.

The coordinates for all graphics commands are the usual 360x200 or 640x200 resolutions available with IBM's color graphics adapter. Strangely, however, some commands specify coordinates (0,0) as the upper left corner, while the TurtleGraphics commands use (0,0) as the screen center. I would have preferred (0,0) to be at bottom left, where my high school teachers told me the origin of an (X,Y) axis should be located.

NAME CHANGES

A few commands and identifiers changed, for reasons not entirely clear. Previously, HighVideo switched on enhanced text on the monochrome display while LowVideo switched on dim characters. In the new version, NormVideo appears to do the same thing as HighVideo, which still works. I don't have any idea why NormVideo was added. HighVideo and LowVideo seem clear enough to me.

More critical are changes to the old Init and Exit procedures. In versions 1.0 and 2.0, Init sent an initialization string to the terminal, while Exit sent a deinitialization sequence. These strings could be programmed with Turbo's installation program, TINST.COM. The procedures are more useful with standard MS-DOS and CP/M versions of Turbo Pascal than they are with the stock PC-DOS, which does not require installation. In the new version, Init no longer exists. Any program using Init fails to compile. The same procedure is now called CrtInit.

Exit, however, does exist but now does something altogether different. To send the deinitializing sequence to the terminal, use CrtExit instead. The new Exit aborts a block of code, which could be a procedure or the main program block.

In other words, if your old programs used Exit, they will now leave the current block when Exit is executed. It's a good idea, therefore, to change Exit to CrtExit and Init to CrtInit as one of your first steps in bringing your library up to date.

WISH LIST

That covers the major differences in the new version, and I wonder what's next for Turbo Pascal? I can't speak for Borland, but my wish list includes the following.

You still cannot pass procedures and functions as parameters to other routines. In all fairness to Borland, few Pascal compilers allow this, probably due to the complexity involved in checking the syntax of parameter lists passed along with passed procedures. The feature would be helpful, however, for writing recursive co-routines, B-Tree search functions, or sorters that call a variety of match functions.

Less understandable is not being able to allocate variant cases of records with NEW and reclaim the space with DISPOSE. Let's say you have the following record definition:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
TYPE
   StringPtr =
      ^RECORD
         CASE StringSize OF
            10 : ( S10 : String[10] );
            20 : ( S20 : String[20] );
            30 : ( S30 : String[30] )
         END;
VAR
   StringRec : StringPtr;
```

Then, in most Pascal's, to allocate memory for one of the variants, you can do this:

```
NEW( StringRec, S20 );
```

That would create a pointer StringRec to the 20-character string variant with tag field S20. In Turbo Pascal, however, you can reserve only the largest variant.

```
NEW( StringRec );
```

Now you can access the smaller variant with StringRec ^.S20, but this wastes the 10 extra bytes allocated by Turbo Pascal to handle the unused case of tag field S30.

If Turbo Pascal did have the ability to allocate space for individual tag fields, then it would also have to allow disposal of those same variants. To dispose the S20 variant, you would then do this:

```
DISPOSE( StringRec, S20 );
```

It would also be nice if we could execute .COM files under program control, passing open files, variables, and the DOS environment, if possible. Originally, Turbo Pascal had a provision for doing this, but it didn't operate in MS/PC-DOS. (It does work in CP/M, however.) Having this feature would allow overlays to be developed in different languages. You could have one part of a program written in assembler, another in C, and other parts in Turbo Pascal.

I'm also one of those people who were sorry to lose Get and Put, and I would be happy to see them reinstated. Despite the probably correct argument that these I/O procedures are dinosaurs best left to sink in the tar pits, there are plenty of published programs that use them — programs that need major surgery to run in Turbo Pascal. Borland would increase its software base if it added Get and Put.

Finally, I would love to see an option to let the compiler generate a list of errors, rather than halting at each one. In a file with six errors, you have to compile your text seven times just to fix the errors and obtain a finished result. I would settle for half the compiler speed if I could generate a list of all errors (even if that meant getting some erroneous errors due to "propagation").

THE LAST STRAW

Alas, when I phoned to suggest some of these things, my call was cheerfully answered, and I spoke to a technical representative who seemed to understand what I was talking about. When Borland wakes up, stops listening to its customers, cuts me off from time to time, and offers rudeness courses to its employees, then perhaps we can welcome Philippe Kahn and company to the real world.

But then, perhaps I'm asking for too much. □

Tom Swan is the author of several computing books, including Mastering Turbo Pascal, *published with Turbo Pascal source code diskettes by Hayden Book Company. His newest book,* Pascal Programs for Data Base Management, *is due out in early 1986. Readers may write directly to the author at P.O. Box 206, Lititz, PA 17543.*

CHANGES?
DEADLINES?
BUGS?
NEED A SMALL MIRACLE?
REFER
• Saves you hours programming.
• Processes any language (ASM, BASIC, C, COBOL, PACAL, ADA, FORTRAN, DBASE, RBASE, ETC.)
• Pretty Prints program source.
• Cross-references concurrently: variables, values, keywords, line numbers and/or labels.
• Fast and flexible utility.
• Source code and examples included.
ONLY $39
JAMES HALSTEAD & ASSOC.
1551 Plainfield, Joliet, IL 60435
(815) 725-0346

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOPVIEW PROGRAMMER'S TOOLKIT

by Ronald Moak

Programmer's ToolKit ($395.00)
IBM Corp., Customer Relations Department,
P.O. Box 1328
Boca Raton, Florida 33432
1-800-447-4700

In August 1984, IBM announced that it would be shipping a new program in the first quarter of 1985 that would allow multi-tasking capabilities on the IBM-PC. After we waited months for TopView (the program that would be the answer to the frustration of waiting for one process to complete before we could start another), it was finally shipped. While TopView has been met with mixed reviews, it does provide the programmer with a rich selection of tools with which to develop applications that take advantage of its multi-tasking and windowing environment.

IBM has broken TopView into the two parts, application and development. This keeps the price of TopView in a range where it is more acceptable to the buyer and separates the technical information for those who wish to write applications that make use of the TopView environment.

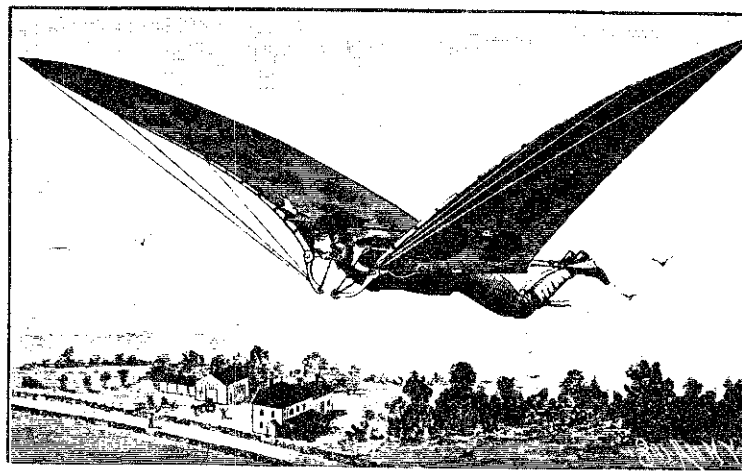
Use of the TopView Programmer's ToolKit requires you to also have a copy of TopView, as several of the programs provided will not run outside the TopView environment. Although the programs may run on a 256K-of-RAM and two-floppy-drive system, any serious application development will require a minimum 512K of RAM and 10 Meg of storage. In general, the latter is the minimum system for running the TopView environment if any serious work is to be performed.

The TopView Programmer's ToolKit is the antithesis of the TopView manual. The TopView applications manual is skimpy and provides only the information needed to run TopView; the tool kit manual is fat (I didn't bother to count the pages) and crammed full of information that could take you months to digest.

The manual is broken into nine chapters plus miscellaneous appendices and covers the following topics:

THE TOP VIEW SYSTEM

This section is an overview of the TopView concepts of windows, multi-tasking, pointing devices (mice), and data transfer between programs.



top view

USING THE INTERFACE

The interface is the assembler interface between the application program and the TopView environment.

USING THE I/O CLASSES

TopView breaks I/O into six different classes. This section covers the calls needed to use the Keyboard, Mailbox, Object, Panel, Pointer, and Timer classes.

USING WINDOWS

This chapter is similar to the last except that it is dedicated to the Window class.

USING THE WINDOW MANAGER

A description of the interface between the windows and the window manager is provided. The window manager performs all of the functions needed to switch programs, change window sizes, etc.

CREATING WINDOWS USING THE WINDOW DESIGN AID

The use of the Window Design Aid program is described. This program allows you to create panels or windows that may be used by the applications program. The panels may be used for full screen editing of data for programs or menus to be called by the program.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

USING THE OTHER PROGRAMMING TOOLS

The other tools include a program to prepare several panels created by the Window Design Aid to be merged into a panel library that may in turn be converted into an object module and linked into the program.

CREATING COMPATIBLE APPLICATIONS

The concepts for creating programs compatible with the TopView environment are discussed.

WRITING A POINTING DEVICE DRIVER

The final chapter discusses the mechanisms for writing your own device drivers to handle mice.

The TopView Programmer's ToolKit comes with two disks in addition to the manual. Disk 1 provides the programs needed to create and manipulate panels, plus a program to create a program information file. This file is used by TopView to integrate the program with the TopView environment. The assembly language listing of TopView's clock program is also provided. The disk also contains a sample program that uses TopView. The program is written in both assembler and Pascal. The Pascal program is provided to illustrate the interface between TopView and a high level language.

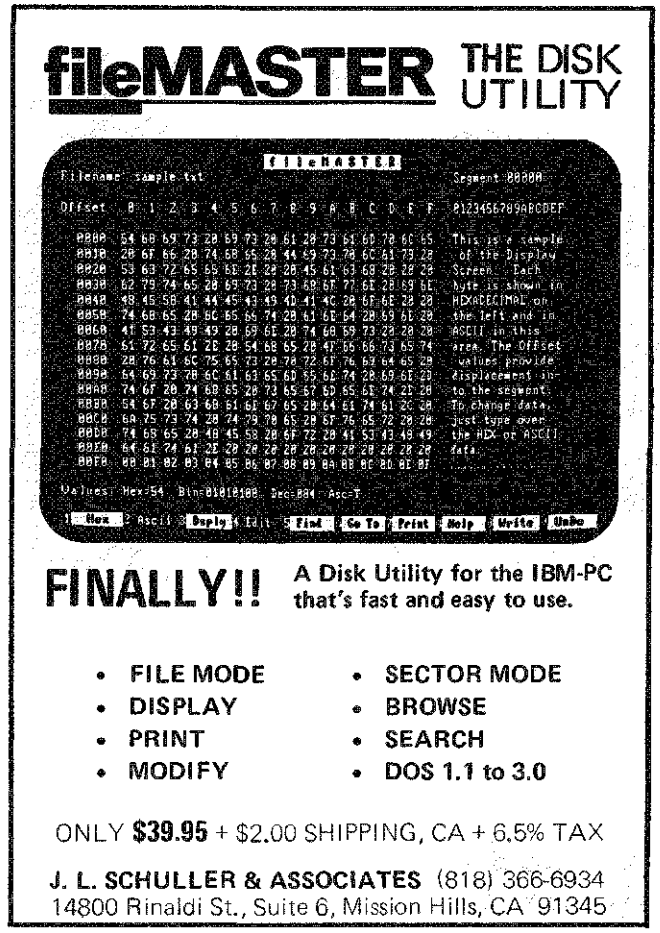
COMPLETE LISTING OF FILES ON DISK 1

CLKALARM	PNL	521	PANELOBJ	PIF	369
CLKALRMH	PNL	493	PDIO	OBJ	3886
CLKDATEH	PNL	339	PTRDEV	INC	1259
CLKHELP	PNL	305	SAMPERR	PNL	281
CLKTIMEH	PNL	340	SAMPFUNC	OVR	108
CLOCK	ASM	61159	SAMPINP	PNL	206
CLOCK	PLB	2071	SAMPLE	EXE	25600
CREATEPF	EXT	10752	SAMPLE	PAS	12215
CREATEPF	HLP	2616	SAMPLE	PIF	369
CREATEPF	PIF	369	SAMPLE2	ASM	20214
INT15	INC	2920	SAMPLE2	COM	1259
INT15	MAC	2570	SAMPLE2	PIF	369
INT15EQU	INC	2900	SAMPOUT	PNL	183
INT15P	ASM	9622	WDA	EXT	64256
INT15P	INC	820	WDA	PIF	369
INT15P	OBJ	455	WDA256	BAT	17
PANELLIB	COM	1933	WDA256	PIF	369
PANELLIB	PIF	369	WDAHELP	PLB	35828
PANELOBJ	COM	2369			

Disk 2 contains a number of files that illustrate the interface between TopView and a high level language. IBM chose Pascal as the language to show off the TopView interface. The files included are a mixture of assembly language modules, panel files, and Pascal source code. A document file describing the functions provided is included.

COMPLETE LISTING OF FILES ON DISK 2

ASMSAMP	ASM	10852	MOVEWNDW	ASM	1721
ASMSAMP	EXE	7424	PASCSAMP	EXE	28416
ASMSAMP	PIF	369	PASCSAMP	PAS	7360
BOTMWNDW	ASM	1368	PASCSAMP	PIF	369
CREATEWN	ASM	2687	PASPROCS	INC	2206
DATAREA	ASM	8454	PUTPANEL	ASM	11665
DRAWWNDW	ASM	1271	PUTPANO	ASM	5973
DROPWNDW	ASM	1458	SAMPERR	PNL	281
FILLFLD	ASM	3406	SAMPFUNC	OVR	108
FILLOUTP	ASM	2798	SAMPINP	PNL	206
FUNCTION	ASM	2626	SAMPOUT	PNL	183
FUNCTION	DOC	24703	SEND3WDS	ASM	1816
FUNCTION	OBJ	8906	SEND3WM	ASM	1894
GETBUTTN	ASM	2922	SETCURRC	ASM	1782
GETCURSR	ASM	1963	SETCURSR	ASM	2158
GETFLD	ASM	4320	SHOWWNDW	ASM	1375
GETMENU	ASM	3516	SOUND	ASM	1075
HIDEWNDW	ASM	1379	TOPWNDW	ASM	1347
INTERNAL	ASM	13621	WAITINPT	ASM	2581
MACROEXT	INC	8348	WRTSTRNG	ASM	2225
MOVEPTR	ASM	2219			


fileMASTER
THE DISK UTILITY
FINALLY!!
A Disk Utility for the IBM-PC that's fast and easy to use.
• FILE MODE
• DISPLAY
• PRINT
• MODIFY
• SECTOR MODE
• BROWSE
• SEARCH
• DOS 1.1 to 3.0
ONLY $39.95 + $2.00 SHIPPING, CA + 6.5% TAX
J. L. SCHULLER & ASSOCIATES (818) 366-6934
14800 Rinaldi St., Suite 6, Mission Hills, CA 91345

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C

HOW EASY DEVELOPING PROGRAMS CAN BE WITH THE C UTILITY LIBRARY!

COMPLETE. Over 300 tested and well documented functions. All features of the PC &AT are at your finger tips.

ADVANCED. Features like windows, data entry fields, switching displays, and batch file execution are supported.

GRAPHICS. A complete set of fast business graphics functions. Our low level graphics routines are the fastest in the business.

SOURCE. All source code is included. 95% of the library is written in C. Only functions demanding fast execution speed are in Assembler.

COMPATIBLE. With Lattice, Microsoft(3.0), CI-C86, Aztec, DeSmet, Wizard, and Mark Williams. All memory models are supported.

MUCH MORE. Dozens of string functions, the best time & date math and formatting, DOS directory and file mgmt., keyboard control, polled async communications, and more. **NO ROYALTIES.** We **SUPPORT** what we sell.

C UTILITY LIBRARY - $185
COMPILERS: Lattice $349, CI-C86 $329, Mark Williams $449. **Save $40-$50** with compiler & library package.

Specify compiler and version number when ordering. Add $4 for UPS or $7 for UPS 2-day shipping. NJ residents add 6% sales tax. VISA, MC, Chks & qualified PO's.

ESI
Essential Software, Inc.
P.O. Box 1003
Maplewood, New Jersey 07040
914/762-6605

CIRCLE NO. 140 ON READER SERVICE CARD

Who should buy this package? The TopView Programmer's ToolKit is geared for people who have a good knowledge of assembler and feel comfortable with it. Even though the Pascal interface is provided, it is there more as a guide to writing your own interface between your language and TopView than as a usable item. It's doubtful that the interface would make any sense without some familiarity with assembler.

It is apparent that at some point IBM or a third party will provide a better interface between the language you are using and TopView. I believe that Computer Innovations already has a C library package that will interface with TopView. Others are sure to follow.

Does all of this mean that if you are not comfortable with assembler you have no use for the package? Certainly not, unless the TopView language interface package you purchase includes its own window design and panel manipulation utilities. These programs can save you hours of work in program development when you're creating input screens and menus.

If you program in assembler and choose to integrate your programs with TopView, then this package is a must. The documentation is much better than many assembler manuals I've tried to read in the past. Numerous examples are available (including the complete source listing of a program on disk) on the disks and in the manual. It may take a while to get used to pictures of mice walking around the pages of a technical manual, but it sure beats the old photocopied CP/M assembler manuals. □

GREAT MOMENTS IN PERSONAL COMPUTER HISTORY: The first Users' group meeting

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BASIC COMPILER

Microsoft has released the QuickBASIC compiler as a companion to their BASIC Interpreter, claiming execution speeds of three to ten times faster than the interpreter. According to Microsoft, users with QuickBASIC can break up large programs into subprograms that can use their own variables or share variables with the main program. Subprograms can be compiled separately and then linked with the main program before it is run. Line numbers are no longer required; lines are executed in the order listed. Alphanumeric labels can designate program sections. Subprograms can be called to return values to the main program. QuickBASIC supports BASIC control structures, including WHILE/WEND, FOR/NEXT, and IF/THEN/ELSE. The new compiler is compatible with the IBM BASICA interpreter to support sound and graphic statements. QuickBASIC is compatible with all versions of BASIC provided with the DOS operating system, including DOS 2.0 and 3.0. It requires 128K of RAM (256 K is recommended), one floppy disk drive (two are recommended), and DOS 2.0 or higher.

Microsoft Corporation
(800) 426-9400

C CHECKER

Gimpel Software has released PC-Lint for diagnosis of C programs running under MS-DOS. PC-Lint will analyze and report on type inconsistencies across modules, parameter-argument mismatches, library usage irregularities, uninitialized variables, value-return inconsistencies, variables declared but not used, suspicious use of operators, and unreachable code. Features include full K&R support, one-pass operation, use of the large memory model internally with all tables expandable. PC-Lint is configurable to arbitrary architectures and uses Lint-style comments to suppress errors. PC-Lint is delivered with user-modifiable standard library descriptions for most major compilers. It requires 128K and will use all the memory available. It can be ordered prepaid for $139 including shipping and handling within the U.S.

Gimpel Software
3207 Hogarth Lane
Collegeville, PA 19426
(215) 584-4261

TURBO POWER AND COMMUNICATIONS

Turbo POWER TOOLS, a series of procedures to complement Turbo Pascal on the IBM PC, is available from Blaise Computing. Features include string handling, screen support and window management, access to BIOS and DOS services, full program control allowing execution of any other program from within Turbo applications, and interrupt service routine support written entirely in Turbo Pascal. Blaise also offers Turbo ASYNCH to allow incorporation of asynchronous communication capabilities into Turbo Pascal application programs. Its features include initialization of the COM ports allowing the setting of all transmission options, interrupt processing, data transfer between circular queues and communications ports, simultaneous bufferd input and output to both COM ports, transmission speeds up to 9600 baud, and XON/XOFF protocol. Turbo POWER TOOLS and Turbo ASYNCH sell for $99.95 each, and additional royalties are not required for their incorporation into end user applications. Each is shipped with a User Reference Manual, all source code, and sample programs.

Blaise Computing, Inc.
2034 Blake Street
Berkeley, CA 94704
(415) 540-5441

CHASM UPGRADED

Whitman Software offers Version 4.0 of CHASM as a total rewrite in Turbo Pascal, running 15 times faster than CHASM Version 2.0 and two-and-one-half times faster than Version 3.0. Other improvements include expanded memory utilization, improved macro facility (including nesting), operand expressions, DOS 2.0 path support for all files, assembly results reported via DOS 2.0 ERRORLEVEL, sorted symbol table dump, improved conditional assembly, and new IF pseudo-ops. CHASM 4.0 requires DOS 2.0, 128K, and one disk drive on computers BIOS-level compatible with the IBM PC. To receive the upgrade at the higher registration price ($40), send Whitman a check for the difference between your original registration payment and $40, your name, address, computer model, and disk format (single- or double-sided). They'll provide the disk and mailer.

Whitman Software
136 Wellington Terrace
Lansdale, PA 19446
(215) 362-8526

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Beginning in the Next Issue of PROGRAMMER'S JOURNAL:

PJ BILLBOARD

An advertising directory listing products and services of interest to the IBM PC programmer

Starting in the Nov./Dec. issue, we will be bringing you the latest in computer products: utilities, hardware, publications and information.
All in **PJ BILLBOARD.**

Note to advertisers:

A listing in **PJ BILLBOARD** brings your product to the attention of the people you want to reach — IBM PC programmers.

Be with us for the very first appearance of this *all new* feature in the Nov./Dec. **Programmer's Journal!**

Your listing of up to 400 characters is only $65.00, prepaid, for each insertion.

For more information on how you can be in the first PJ BILLBOARD, just call:

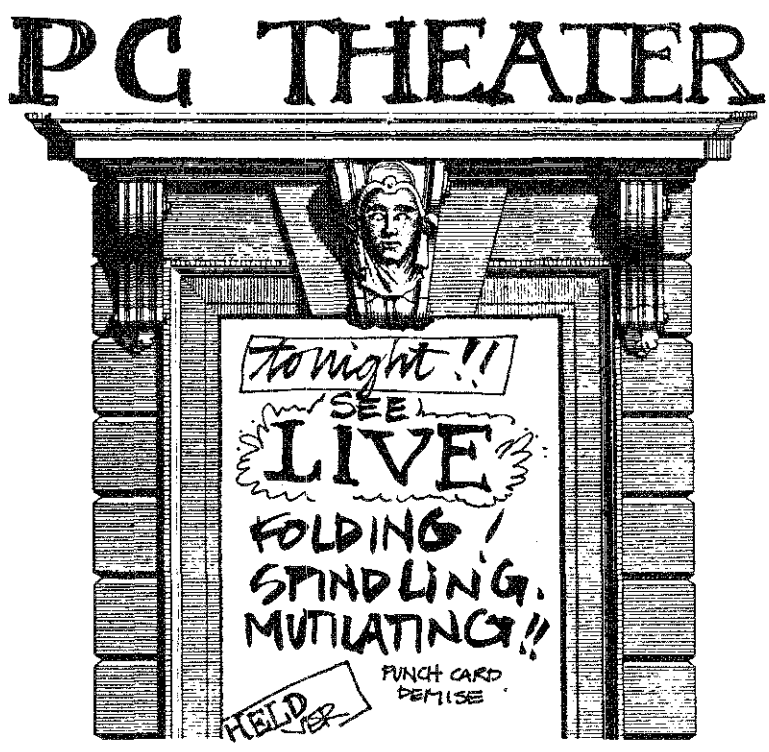
503-484-0602

or send your typewritten copy, along with a check to:

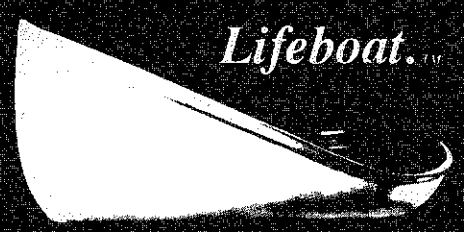
PJ BILLBOARD
P.O. Box 30160
Eugene, OR 97403

ADVERTISERS INDEX

Blaise Computing . . . . . Cover/Inside Front
Borland International . . . . . Cover/Back
Computer Faire, Inc. . . . . Cover/Inside Back
Cybernetic Micro Systems 9
Data Base Decisions 31
Datalight 15
Ecosoft, Inc. 8
Essential Software 44
Gimple Software 37
Guidance Software 19
Greenleaf Software 37
Hawkeye Grafix 36
James Halstead Associates 41
J. L. Schuller & Associates 43
Landmark Software 29
Lifeboat Associates 47
OES Systems 30
OPT-Tech Data Processing 6
Phoenix Computer Products 5
Programmers Shop 24
Qualitas 36
Rational Systems, Inc. 11
Software Horizons 11
Sterling Castle 5



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



C is the language.
Lifeboat™ is the source.

Productivity Tools from the Leading Publisher of C Programs.™

The Lattice® C Compiler

The cornerstone of a program is its compiler; it can make the difference between a good program and a great one. The Lattice C compiler features:

- Full compatibility with Kernighan and Ritchie's standards
- Four memory model options for control and versatility
- Automatic sensing and use of the 8087 math chip
- Choose from the widest selection of add-on options
- Renowned for speed and code quality
- Superior quality documentation

"Lattice C produces remarkable code...the documentation sets such a high standard that others don't even come close...in the top category for its quick compilation and execution time and consistent reliability."

Ralph A. Phraner, *Byte* Magazine

Lattice Library source code also available.

Language Utilities

Pfix 86/Pfix 86 Plus — dynamic and symbolic debuggers respectively, these provide multiple-window debugging with breakpointing capability.
Plink 86 — a two-pass overlay linkage editor that helps solve memory problems.
Text Management Utilities — includes GREP (searches files for patterns), DIFF (differential text file comparator), and more.
LMK (UNIX "make") — automates the construction of large multi-module products.
Curses — lets you write programs with full screen output transportable among all UNIX, XENIX and PC-DOS systems without changing your source code.
BASTOC — translates MBASIC or CBASIC source code directly to Lattice C source code.
C Cross Reference Generator — examines your C source modules and produces a listing of each symbol and where it is referenced.

Editors

Pmate — a customizable full screen text editor featuring its own powerful macro command language.
ES/P for C — C program entry with automatic syntax checking and formatting.
VEDIT — an easy-to-use word processor for use with V-PRINT.
V-PRINT — a print formatting companion for VEDIT.
CVUE — a full-screen editor that offers an easy way to use command structure.
EMACS — a full screen multi window text editor.
Fast/C — speeds up the cycle of edit-compile-debug-edit-recompile.

Graphics and Screen Design

HALO — one of the industry's standard graphics development packages. Over 150 graphics commands including line, arc, box, circle and ellipse primitives. The **10 Fontpack** is also available.
Panel — a screen formatter and data entry aid.
Lattice Window — a library of subroutines allowing design of windows.

Functions

C-Food Smorgasbord — a tasty selection of utility functions for Lattice C programmers; includes a binary coded decimal arithmetic package, level 0 I/O functions, a Terminal Independence Package, and more.
Float-87 — supports the 8087 math chip to boost the speed of floating-point calculations.
The Greenleaf Functions — a comprehensive library of over 200 routines.
The Greenleaf Comm Library — an easy-to-use asynchronous communications library.
C Power Packs — sets of functions useful for a wide variety of applications.
BASIC C — This library is a simple bridge from IBM BASIC to C.

Database Record Managers

Phact — a database record manager library of C language functions, used in the creation and manipulation of large and small databases.
Btrieve — a sophisticated file management system designed for developing applications under PC-DOS. Data can be instantly retrieved by key value.
FABS — a Fast Access Btree Structure function library designed for rapid, keyed access to data files using multipath structures.
Autosort — a fast sort/merge utility.
Lattice dB-C ISAM — a library of C functions that enables you to create and access dBase format database files.

Cross-Compilers

For programmers active in both micro and mini environments we provide advanced cross-compilers which product Intel 8086 object modules. All were developed to be as functional — and reliable — as the native compilers. They are available for the following systems:

VAX/VMS, VAX/UNIX, 68K/UNIX-S, 68K/UNIX-L

Also, we have available:
Z80 Cross-Compiler for MS- and PC-DOS — produces Z80 object modules in the Microsoft relocatable format.

New Products

Run/C — finally, a C interpreter for all levels of C Programmers.
C Sprite — a symbolic debugger with breakpoint capability.

Call LIFEBOAT: 1-800-847-7078. In NY, 1-212-860-0300.

YES! Please rush me the latest FREE Lifeboat™ catalog of C products.

*Name*_____ *Title*_____

*Company Name*_____ *Business Phone*_____

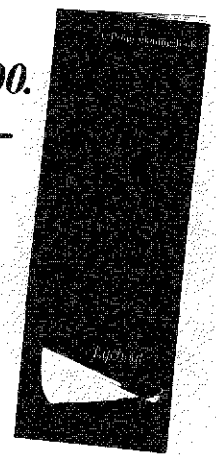
*Address*_____

Please check one of the following categories:

☐ *Dealer/Distributor* ☐ *End User* ☐ *Other*_____

Return Coupon to: Lifeboat™ Associates
1651 Third Avenue, New York, NY 10128

© 1985 Lifeboat Associates



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ARE YOU AN ISSUE SHY OF A FULL SET?

- Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.
- Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
- Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.
- Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILEWNED, C/ASM interface, ASM stay resident routine, Structured programming.
- Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBase bargraphs, Utilities.
- Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi- speed graphics, Software ownership, dWINDOW review.
- Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing national database, Ultra utilities.
- Vol. 2, No 2: Understanding C through Assembly language, Utility routines in C and Assembly, Music in C (using a C music program), Cornucopia of software development tools.
- Vol. 2, No. 3: Software development environments, Creating a productive screen handling tool, Generating graphics within windows, Create, edit and play your musical ideas, Linking Assembler to Pascal and Basic.
- Vol. 2, No. 4: Writing a custom ROM BIOS for PC-DOS, The future of disk protection, Time-saving MASM Macros, C routines you can purchase.
- Vol. 2, No. 5: Scroll control using BASIC and Assembler, RTool, a PC-resident programmer's toolkit, Calling Assembly routines from dBase II, Pascal's pointer power.
- Vol. 2, No. 6: Periscope—Building a hardware board for the IBM PC, AT&T surprises all, dBase II to dBase III?, Intel 80286 bugs by FNE, calendar functions for BASIC.
- Vol. 3, No. 1: November's Comdex, C—its uses and abuses, Using the C preprocessor, Memory segmentation examples, Index files (part 1), Structured programming language, Periscope, a symbolic debugger.
- Vol. 3, No. 2: UNIX! (again), Artificial intelligence, Doctor C's tips — pointers to functions, Calendar date routines in BASIC, Calculate record layouts, automatically!
- Vol. 3, No. 3: NAPLPS and the professional programmer, Instant-C, Printer control from a batch file, Repositioning the stars in Starfinder, Reducing EXE file size through dtat positioning, tools for better programming.
- Vol. 3, No. 4: More UNIX stuff, C programming books, C code excerpts, How to cure the 'dreaded locked keyboard' condition, Calling MS-DOS from Turbo Pascal, DOS 2.X sort and find filters, the SET Tools.

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone _____

Send me the following back issues: (Circle the issue(s) desired)
$5.50 each U.S., ($6.50 Canada, $8.50 Foreign, U.S. funds)
PRE-PAID ONLY

Vol. 1, No. 1	Vol. 1, No. 5	Vol. 2. No. 3	Vol. 3, No. 1
Vol. 1, No. 2	Vol. 1, No. 6	Vol. 2. No. 4	Vol. 3, No.2
Vol. 1, No. 3	Vol. 2, No. 1	Vol. 2. No. 5	Vol. 3, No. 3
Vol. 1, No. 4	Vol. 2, No. 2	Vol. 2. No. 6	Vol. 3, No. 4

Total amount of order _____

☐ Check enclosed

☐ Charge my Bankcard: ☐ VISA ☐ MC

_____ Expir. Date _____

Signature _____

Send to:
PROGRAMMER'S JOURNAL
P.O. Box 30160
Eugene, OR 97403
(503) 484-2162

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Catch "The Wave" This Fall!

A massive wave of microcomputer products, services and accessories will crest in Boston this fall, as **The 8th Northeast Computer Faire & UNIX* Systems Expo** floods the Bayside Exposition Center, September 26–29, with:

- **Thousands of Computer Products** exhibited by hundreds of leading and innovative companies, including IBM, AT&T, and many others.
- **Special Areas For "Hands-On" Demos.**
- **A Comprehensive Conference Program** (included in admission) attracting industry speakers and computer users from business, education, government and scientific, engineering & research communities.
- **Professional Development Seminars.**
- **Free One-On-One Consultations.**
- **Incredible "Show Pricing"** on all kinds of products.

Ride this giant wave of microcomputer products, services and accessories at **The 8th Northeast Computer Faire & UNIX* Systems Expo** this fall!

*UNIX is a trademark of AT&T Bell Laboratories

Keynote: Software entrepreneur **Philippe Kahn,** president of **Borland International,** "a software success story."

Computer Clinics: sponsored by the Independent Computer Consultants Association. Bring your questions and get answers to your software problems.

IBM Technical Seminars – Free seminars for software developers sponsored by IBM.

Conference Session Highlights:

Artificial Intelligence and Expert Systems
The Coming Explosion in Optical Disc Technology
The MSXplosion!™
UPDATE: Fifth Generation Computing Projects Worldwide
Regulatory Issues: Exporting High Technology Products
How to Choose a Small Business System
Upgrading Your Personal Computer

An exclusive production of **Computer Faire, Inc.**/A Prentice-Hall Company

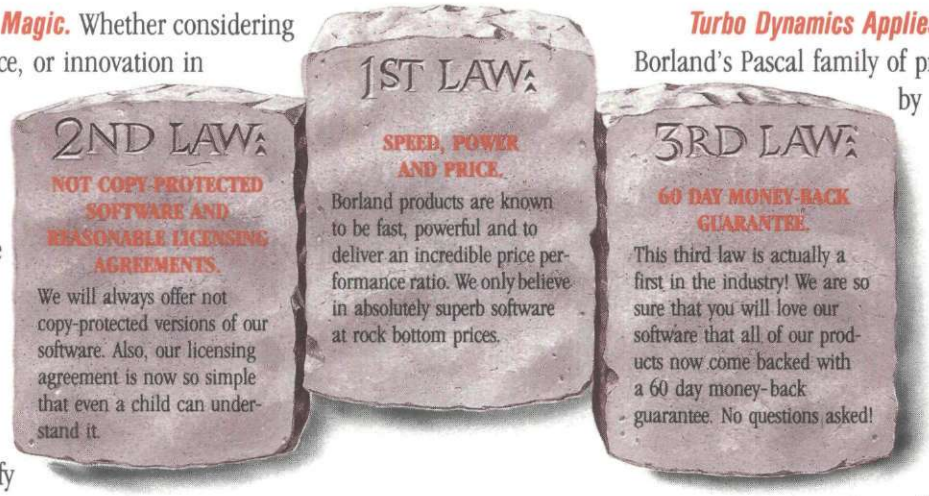
181 Wells Avenue
Newton, MA 02159
617/965-8350

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PC WEEK'S PRODUCT OF THE YEAR
PC MAGAZINE'S AWARD FOR TECHNICAL EXCELLENCE

Borland Introduces the Laws of *TURBO DYNAMICS*™

Laws That Work Like Magic. Whether considering technological excellence, or innovation in areas such as pricing, not copy-protection, licensing agreements, site licenses, 60 day money-back guarantee —Borland is clearly recognized as the software industry leader. The following three laws of "*Turbo Dynamics*"™ exemplify our pledge for excellence.


2ND LAW:
NOT COPY-PROTECTED SOFTWARE AND REASONABLE LICENSING AGREEMENTS.
We will always offer not copy-protected versions of our software. Also, our licensing agreement is now so simple that even a child can understand it.
1ST LAW:
SPEED, POWER AND PRICE.
Borland products are known to be fast, powerful and to deliver an incredible price performance ratio. We only believe in absolutely superb software at rock bottom prices.
3RD LAW:
60 DAY MONEY-BACK GUARANTEE.
This third law is actually a first in the industry! We are so sure that you will love our software that all of our products now come backed with a 60 day money-back guarantee. No questions asked!

Turbo Dynamics Applies to Turbo Pascal. Borland's Pascal family of products is growing by leaps and bounds. You can now join hundreds of thousands of users and enter the world of Turbo Pascal programming. And remember, all three laws of *Turbo Dynamics* apply to all Borland products.

TURBO PASCAL™ $69.95

The industry standard. With more than 350,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use. **Free spreadsheet** included on every Turbo disk with ready-to-compile source code. ***Options:*** We offer the exciting Binary Coded Decimal (BCD) option for your business applications as well as an 8087 option for your number-crunching applications at a very low charge. Please refer to the coupon. ***Portability.*** Turbo Pascal is available today for most computers running PC-DOS, MS-DOS, CP/M-80 or CP/M-86. ***Jeff Duntemann, PC Magazine:*** "In its simplicity it achieves an elegance that no other language compiler has ever displayed."

TURBO GRAPHIX TOOLBOX™ $54.95

High resolution monochrome graphics and window management for the IBM PC. The Turbo Graphix Toolbox will give even a beginning programmer the expert's edge. It's a complete library of Pascal procedures and functions. Tools that will allow you to draw and hatch pie charts, bar charts, circles, rectangles and a full range of geometric shapes. Procedures that will save and restore graphic images to and from disk. And much, much, more. You may incorporate part or all of these tools in your programs and yet we won't charge you any royalties. Best of all, these functions and procedures come complete with commented source code on disk ready to compile.

TURBO TUTOR™ $34.95

From start to finish in 300 pages. Turbo Tutor is for everyone from novice to expert. Even if you've never programmed before Turbo Tutor will get you started right away. ***A must.*** You'll find the source code for all the examples in the book on the accompanying disk ready to compile. Turbo Tutor might be the only reference on Pascal and programming you'll ever need.

TURBO DATABASE TOOLBOX™ $54.95

The Turbo Database Toolbox (formerly Turbo Toolbox) is the perfect complement to Turbo Pascal. It contains a complete library of Pascal procedures that allows you to sort and search your data and build powerful applications. It's another Borland set of tools that will give the beginning programmer the expert's edge. ***Get started right away: free database!*** Included on every Toolbox disk is the source code to a working database which demonstrates how powerful and easy to use our search system, Turbo-Access, really is. Modify it to suit your individual needs or just compile it and run. ***Remember, no royalties!***

NOT COPY-PROTECTED

TURBOPASCAL FAMILY

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

Carefully Describe your Computer System!

Mine is: ___ 8 bit ___ 16 bit
I Use: ___ PC-DOS ___ MS-DOS
___ CP/M 80 ___ CP/M 86

My computers' name/model is: _____

The disk size I use is:
☐ 3½" ☐ 5¼" ☐ 8"

60 DAY MONEY-BACK GUARANTEE

Pascal 3.0	$ 69.95
Pascal w/8087	$109.90
Pascal w/BCD	$109.90
Pascal w/8087 & BCD	$124.95
Turbo Toolbox	$ 54.95
Turbo Graphix	$ 54.95
Turbo Tutor	$ 34.95

*These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

Amount: (CA 6% tax) _____
Payment: VISA MC BankDraft Check
Credit Card Expir. Date: ___/___
Card #: _____

Name: _____
Shipping Address: _____
City: _____
State: _____ Zip: _____
Telephone: _____

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

F36

BORLAND INTERNATIONAL
4585 Scotts Valley Drive, Scotts Valley CA 95066
Phone (408) 438-8400 Telex 172373

Copyright 1985 Borland International BI-1011

Turbo Pascal, Turbo Database Toolbox, Turbo Graphix Toolbox, Turbo Tutor and Turbo Dynamics are trademarks of Borland International, Inc.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED